**Universidad Carlos III de Madrid**

Escuela politécnica superior


Ingeniería Informática

Proyecto Fin de Carrera

# Sistema de control colaborativo de robots autónomos basado en planificación automática

**Autor**: Moisés Martínez Muñoz

**Tutor**: Fernando Fernández Rebollo

¿Serán los robots los herederos de la Tierra?

Sí, pero serán hijos nuestros.

Marvin Minsky, 1994

# Agradecimientos

En primer lugar quería agradecer a mi tutor Fernando Fernández, por la oportunidad de realizar este proyecto, así como su ayuda e infinita paciencia a lo largo del proceso de desarrollo de este trabajo.

Al profesor Raúl Arrabales de la Universidad Carlos III, por su ayuda en las etapas iniciales de este proyecto, durante la instalación de los servicios de gestión del sonar, desarrollados por el.

A Susana y David, por acompañarme siempre a lo largo de todo este proceso y apoyarme en todo momento.

A mis compañeros del laboratorio, por su compresión a la hora de rodear el edificio durante la realización de las pruebas y por sus comentarios, los cuales me han servido para mejorar algunos de los elementos de este proyecto.

A todos mis amigos, por aguantar mis continuos comentarios sobre este proyecto y siempre tener tiempo para ayudarme a desconectar.

# Resumen

En este documento se presenta el diseño de una arquitectura de control híbrida, distribuída y modular para robots heterogéneos basada en planificación automática. Su principal objetivo consiste en resolver problemas mediante la actuación combinada de varios robots, los cuales a través de la realización de tareas sencillas, sean capaces de producir comportamientos que puedan ser considerados inteligentes. Para la realización de este proyecto se ha utilizado Microsoft Robotic Developer Studio 2008 R2 como *framework* de trabajo debido al amplio número de módulos y servicios orientados a la robótica que posee, y los robot P3-DX y Lego NXT, para comprobar el correcto funcionamiento de la arquitectura de control que ha sido desarrollada.

# Índice general

# Índice de figuras

# Índice de tablas

# Capítulo 1

# Introducción

La robótica inteligente es un campo de investigación perteneciente a la inteligencia artificial, que se encarga de diseñar y construir sistemas de control para agentes físicos de tipo automáticos, semi-automáticos o teleoperados, mediante la utilización de técnicas basadas en inteligencia artificial. Este proyecto presenta el proceso de desarrollo que un sistema de control híbrido y distribuido para la realización de tareas mediante la colaboración de múltiples agentes físicos independientes.

Utilizando un razonamiento muy básico, podríamos dividir la inteligencia humana en dos niveles. El primero de ellos, al que denominaremos **nivel superior**, es aquel que utilizamos los seres humanos de manera consciente y que nos permite razonar, resolver problemas, por ejemplo de tipo matemático, y que nos dota de creatividad. En cambio el segundo de ellos, el **nivel inferior** es aquel que actúa de manera inconsciente o automática y que nos dota de percepciones, reflejos, control motriz y otra serie de procesos de naturaleza reactiva.

De forma similar podrían construirse los procesos de razonamiento de los robots, utilizando dos niveles de abstracción, uno muy básico que se encargara de producir comportamiento de bajo nivel, los cuales deberían estar programados previamente y serían los comportamientos innatos, y otro de alto nivel, donde los comportamientos deberían producirse mediante la combinación de los de bajo nivel. De esta forma para un robot, la demostración de inteligencia racional sería

la combinación ordenada de los distintos comportamientos de bajo nivel que tiene disponibles, que producen la aparición de comportamiento complejos. Este proceso de generación de conjuntos de comportamientos básicos podría realizarse mediante la utilización, de las ya antes mencionadas, técnicas de **inteligencia artificial**.

Teniendo en cuenta que nuestra intención consiste en construir un sistema que muestre comportamientos complejos de tipo colaborativo, mediante la combinación de procesos muy sencillos, una técnica de inteligencia artificial que se adapta de manera perfecta, es la **planificación automática**. Esta ya ha sido utilizada anteriormente, obteniéndose muy buenos resultados, para la definición de planes de actividades para los *rovers* enviados a Marte por la NASA [1], o en la coordinación de sistemas para la resolución de crisis ambientales mediante el sistema SIADEX [2].

Además, los últimos avances a nivel tecnológico nos presentan un amplio abanico de posibilidades a la hora de intentar simular este tipo de procesos, ya que es posible utilizar sobre los robots actuales técnicas de inteligencia artificial de manera más eficiente. Además, debido a la gran diversidad y versatilidad de robots que existen actualmente es posible incluso construir sistemas colaborativos entre los robots de forma que puedan llevar a cabo tareas conjuntas mucho más complejas mediante la realización de tareas sencillas de forma individual.

Teniendo en cuenta estos factores, la aplicación de sistemas de robots coordinados es viable para la realización de un gran número de tareas que bien por su peligrosidad o por su falta de importancia no son realizadas por los seres humanos.

- Búsqueda y rescate de personas en catástrofes naturales.
- Sistemas de limpieza automatizados de fachadas y/o carreteras.
- Sistemas de gestión automática de almacenes y zonas de carga y descarga.
- Sistemas de exploración de zonas cuyas condiciones ambientales son dañinas para los seres humanos.

Un ejemplo de robot que ha sido desarrollado con un propósito similar a los mencionados es el robot **Vecna Bear**, que se muestra en la figura 1.1. Este robot ha sido desarrollado por la empresa Vecna Robotics, con el fin de evitar que los humanos arriesguen sus vidas en misiones de salvamento o búsqueda de personas.

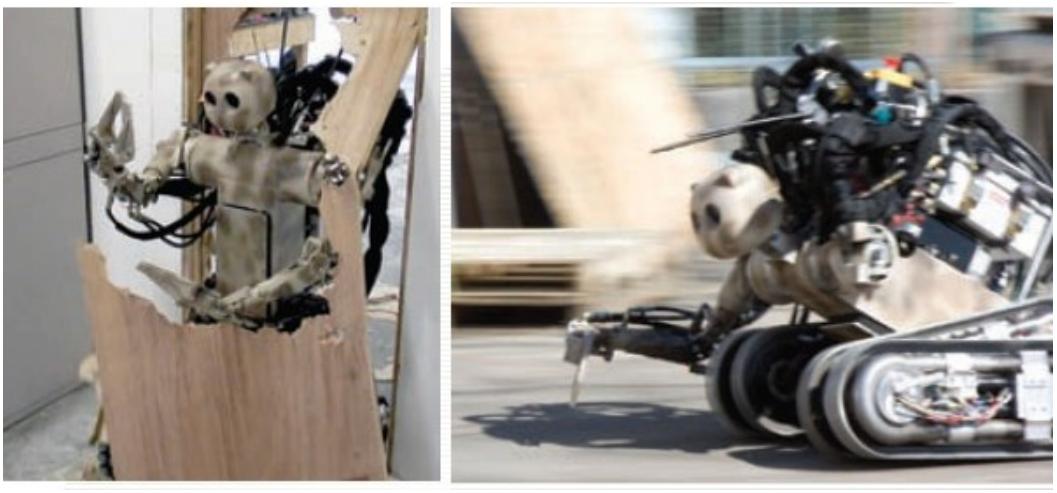

Figura 1.1: Ejemplo de un robot de rescate

## 1.1. Descripción del problema

Actualmente existen situaciones en las cuales los seres humanos no son capaces de realizar ciertas tareas o trabajos. Esto puede ser debido a que no es posible acceder a la zona en la cual deben realizarse, o por que existen ciertos peligros que impiden la realización de las tareas. Mediante este proyecto se presenta el desarrollo de una arquitectura de control mediante la cual un conjunto de robots puedan colaborar para realizar tareas complejas de forma automática, mediante la utilización de técnicas de la inteligencia artificial. De forma más concreta, la estructura de uno de los posibles problemas que se pretenden resolver podría ser la siguiente.

> *Dado un entorno inaccesible para el ser humano, en el cual se dispone de un conjunto de robots de dos tipos, excavadores y transportadores,*

*que son capaces de recoger muestras de los elementos del entorno. Dado este escenario la misión a realizar consiste en recoger muestras de la zona y transportarlas a una zona accesible para el ser humanos, de forma que los robots deberán coordinarse para recoger las distintas muestras y transportarlas a una zona en la cual puedan ser recogidas posteriormente.*

Por lo tanto, el sistema de control a desarrollar debe ser capaz de definir de forma automática las acciones que tienen que ser realizadas por los distintos robots para cumplir con los objetivos planteados. Para poder ofrecer este tipo de comportamientos, se ha seleccionado la planificación automática y se ha tomado el dominio de los *rovers* como punto de partida para definir el dominio que será finalmente utilizado.

### 1.1.1. Dispositivos

Para la realización de este proyecto se han seleccionado dos tipos de robots. A continuación para cada uno de ellos se realiza una descripción detallada de sus características.

#### 1.1.1.1. Robot P3-DX

El primero de los robots utilizados en este proyecto ha sido el robot Pionner P3-DX, presentado en la figura 1.2. Este dispositivo, es un robot de tipo rover, compuesto por un sistema de tracción diferencial, formado por dos ruedas motrices de 19 cm dispuesta lateralmente y una rueda de tipo castor o giro libre situada en la parte inferior del robot que le facilita los operaciones de giro. Para la obtención de información del entorno el robot ofrece dos sistema de sensado, uno formado por un conjunto de 10 sensores de presión (*Bumpers*), distribuidos entre la parte delantera y trasera del robot en grupos de 5. Otro formado por un anillo de ultrasonidos (Sónar) compuesto por 8 transductores, dispuestos a lo largo de 180 grados y separados entre sí cada 15 grados, este se encuentra colocado en la parte delantera

Figura 1.2: Imagen del robot P3-DX

del robot de forma que sólo podemos obtener información de los objetos que se encuentran delante del robot. La disposición de estos elementos se corresponde con las imágenes presentadas en la figura 1.3.

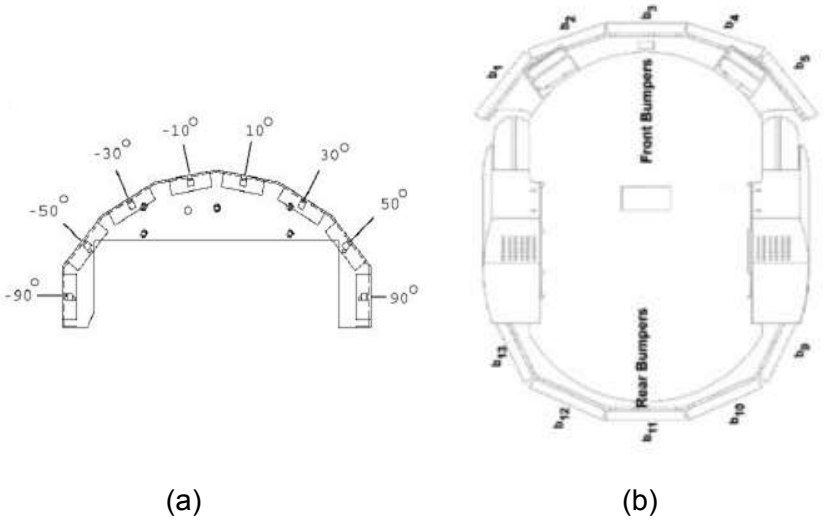


Figura 1.3: (a) Disposición del anillo de ultrasonidos. (b) Disposición de los sensores de presión

Para poder ejecutar el sistema de control desarrollado por el robot, se ha

utilizado un ordenador portátil que ha sido colocado en la parte superior de este, debido a que este no posee un ordenador de abordo propio. Esto supuso la reducción el espacio útil para colocar otros dispositivos o elementos sobre el robot, como en el caso de este proyecto una zona de carga para depositar los objetos que van a ser cargados por el brazo robótico.

#### 1.1.1.2. Robot Lego NXT

Figura 1.4: Imagen del brazo robótico utilizado

El segundo de los robot que va ser utilizado en este proyecto, es un brazo robótico, construido mediante el kit de desarrollo de robot MindsStorm NXT, presentado en la imagen 1.4. Este robot ha sido construido desde cero, mediante la utilización de dos kits de montaje Lego Mindstorm NXT, los cuales permiten la creación de robots sencillos de pequeño tamaño, que nos permiten desplegar el sistema de control.

El robot que ha sido construido está formado por una base sobre la cual el robot puede girar sobre el eje X, y dos eslabones que permiten al robot aumentar su espacio de actuación, en el último de los eslabones se ha incluido una pinza de baja precisión, formada por cuatro garras, con la cual es posible recoger objetos de pequeño tamaño y peso. Para conseguir los 3 movimientos que permite el


Figura 1.5: (a) Actuador servo NXT. (b) Sensor de presión NXT.

brazo robótico desarrollado han sido utilizados tres *servos*, como el presentado en la figura 1.5 y un sensor de presión que permite detectar cuando un objeto es recogido por las pinzas del robot. Este sensor de presión no es muy preciso debido a que dependiendo del peso y la forma del objeto se mantendrá pulsado mientras el objeto se encuentre entre las pinzas o sólo será pulsado cuando el objeto sea recogido.

## 1.2. Motivación

La utilización de agentes físicos o robots para la realización de tareas del ámbito general humano es una realidad, ya que en los últimos años han aparecido un alto número de nuevos dispositivos los cuales son capaces de realizar tareas complejas desarrolladas por los humanos, aunque en muchos casos estas tareas son programadas previamente en los robots. Además, cabe destacar que la evolución tecnológica humana nos brinda cada día nuevas situaciones, en las cuales la utilización de los robots es una posibilidad o incluso en muchas ocasiones un necesidad. Esto implica que es necesario construir sistemas de control que sean capaces de gestionar los procesos de realización de tareas complejas mediante la utilización de inteligencia artificial, de forma que el robot sea capaz de interactuar con los entornos humanos, los cuales poseen un elevado grado de volatilidad. A continuación se describen de manera más detallada las motivaciones que me han

llevado a realizar este trabajo:

- Actualmente existen un gran número de tareas colaborativas en las cuales es posible utilizar conjuntos de robots, debido a la complejidad de las tareas que deben ser realizadas o su peligrosidad, como por ejemplo la extracción de minerales en zonas con un alto grado de peligrosidad, o la realización de futuras prospecciones en otros planetas. En estas situaciones tal vez no es posible usar sistemas teleoperados debido a los posibles problemas de comunicación, por lo que utilizar sistema basados en inteligencia artificial que puedan adaptarse a los posibles cambios que puedan producirse en el entorno es una solución bastante factible.

- Los sistemas organizativos humanos se basan, normalmente, en la organización alrededor de una persona la cual define el trabajo de los demás. Este tipo de organización centralizada podría producirse también en un conjunto de robots que colaboran entre sí para conseguir un objetivo complejo, entre los cuales existe uno de ellos con un sistema de gran potencia computacional, el coordinador, que se encargará de la definición de las tareas generales y de la coordinación de los robots, que ejecutan las acciones recibidas desde el coordinador teniendo en cuenta los cambios que se producen en el entorno.

- La utilización de la planificación automática para la definición de los procesos de razonamiento me permitirá analizar de forma mucho más detallada esta técnica de la inteligencia artificial y comprobar si es posible utilizarla para generar soluciones que permitan la interacción de múltiples agentes físicos, o la ejecución de forma simultánea de acciones por parte de agentes independientes.

Este conjunto de motivaciones da lugar a los distintos objetivos de este proyecto, los cuales son descritos en el siguiente apartado.

## 1.3. Objetivos del proyecto

A continuación se presentan de forma detallada los distintos objetivos que conforman el alcance del presente proyecto.

1. Estudio y análisis de Microsoft Robotic Developer Studio

   Para el desarrollo del sistema de control se ha seleccionado el *framework* de desarrollo Microsoft Robotic Developer Studio 2008 R2. Esto implica que será necesaria la realización de un estudio de las distintas funcionalidades ofrecidas por este *framework*, con el fin de poder utilizarlas en el desarrollo y simplificar el proceso de construcción del sistema de control.

2. Diseño e implementación del sistema de control distribuido

   - Estudio y análisis de trabajos previos

     Con el fin de conseguir construir un buen sistema, será necesario realizar una lectura y compresión de la literatura (artículos, libros y manuales) relacionada con el diseño e implementación de controladores para agentes físicos y la planificación automática.

   - Construcción del robot Lego NXT

     Con el fin de poder utilizar robots de distinto tipo y comprobar el correcto funcionamiento del sistema con control distribuido con varios tipos de robots, ha sido seleccionado el Kit Lego MindStorm 2.0, con el cual se intentará construir un brazo robótico con 2 grados de libertad que pueda ser integrado en la arquitectura que se pretende desarrollar.

   - Diseño de la arquitectura

     Definición de las características básicas de la arquitectura a desarrollar, teniendo en cuentas las ventajas y desventajas de la tecnología que ha sido elegida para desarrollarla.

   - Implementación de la arquitectura de control

     Tras la selección y/o construcción de los distintos robots, será necesario

implementar los distintos elementos que formen parte de la arquitectura que ha sido diseñada.

3. Experimentación y evaluación del sistema desarrollado

   - Diseño de experimentos

     Inserción de modificaciones en el dominio de los rovers y creación de un conjunto de problemas que pudieran ser resueltos mediante los distintos robots conectados al sistema.

   - Realización de experimentos

     Realización de los distintos experimentos diseñados, variando la información desconocida por parte del sistema de control.

   - Análisis de los resultados

     Análisis de los resultados obtenidas tras el proceso de experimentación y reflexión sobre las conclusiones y las posibles lineas futuras de investigación derivadas del desarrollo de este proyecto.

4. Desarrollo de la documentación

   Desarrollo de la documentación descriptiva del proceso de desarrollo del presente proyecto.

## 1.4. Estructura del documento

Este documento se divide en 7 capítulos, de los cuales el último alberga los distintos anexos del proyecto. A continuación se realiza una descripción del contenido de cada uno de ellos:

- El primer capítulo del documento presenta una breve introducción del proyecto, así como las distintas motivaciones que me han llevado a su realización de este proyecto y los distintos objetivos que que fueron planteados al inicio del mismo.

- En el capítulo 2 se presenta el marco teórico, donde se realiza una rápida descripción de la historia de la robótica, pasando por la descripción de las distintas arquitecturas de control robótico que han sido diseñadas para ser utilizadas en el desarrollo de sistemas de control para agentes físicos. En este capítulo se presentan las arquitecturas de control tradicionales como son las aproximaciones deliberativas y reactivas, incluyéndose algunos ejemplos de estas, y las arquitecturas híbridas que intentan aunar las ventajas de las anteriores. Para finalizar se realiza una descripción detallada de la planificación automática la cual tiene una gran importancia en este proyecto debido a que es la técnica de inteligencia artificial aplicada en la sistema de control desarrollado de este proyecto.

- En el tercer capítulo del documento se describe la estructura de la arquitectura desarrollada, definiéndose todos sus elementos de manera detallada. Al final de este capitulo se realiza una valoración de la arquitectura presentada destacando sus ventajas e inconvenientes.

- En el cuarto capítulo se realiza una descripción gráfica de las pruebas y el entorno en el cual fueran realizadas. Teniendo en cuenta la naturaleza de este proyecto en este capítulo no se presentarán resultados empíricos de gran importancia, sino sólo una descripción detallada de las pruebas que han sido realizadas, así como una valoración de las mismas.

- En el capítulo 5 de este documento se presenta la información referente a la planificación del desarrollo de este proyecto así como los costes necesarios para poder llevarlo a cabo.

- En el sexto capítulo se realiza una descripción de los distintos problemas que han surgido durante el proceso de desarrollo, así como las conclusiones que han sido obtenidas tras la realización de este proyecto y las posibles lineas futuras de investigación abiertas tras el desarrollo.

- Al final de este documento se presentan un conjunto de anexos en los cuales

se amplia la información referente a este proyecto, incluyendo una descripción de las distintas tecnologías o *frameworks* que han sido utilizados durante el proyecto, el diseño detallado del sistema de control, el código PDDL de los experimentos realizados, el glosario de términos, etc.

# Capítulo 2

# Estado de la Cuestión

La palabra robótica apareció por primera vez en el relato Runaround en 1942 del escritor Isaac Asimov [3] y puede definirse como la ciencia o rama de la tecnología, que estudia el diseño y construcción de máquinas y/o agentes físicos, capaces de realizar tareas de forma automática sin la intervención de los seres humanos, en muchos casos mediante la utilización de inteligencia artificial.

## 2.1. Historia de la robótica

Durante siglos el ser humano ha intentado crear artefactos que imitaran a los seres vivos o realizaran tareas de forma automática. Ya en la Grecia Clásica surgieron los primeros inventos mecánicos capaces de realizar tareas de forma automática, siendo considerado el primero del que se tiene constancia el desarrollado por el matemático griego Arquitas de Tarento, el cual construyo un pájaro mecánico denominado "la paloma"propulsado mediante vapor y que intentaba imitar el vuelo de las aves.

Durante los siguientes siglos fueron construidos distintos autómatas de mayor o menor sofisticación que intentaban en muchos casos imitar ciertos comportamientos de los seres humanos o de los animales.

- En 1352 se instaló en la catedral de Estrasburgo un autómata con forma de gallo, el cual era capaz de mover el pico y las alas al dar las horas.

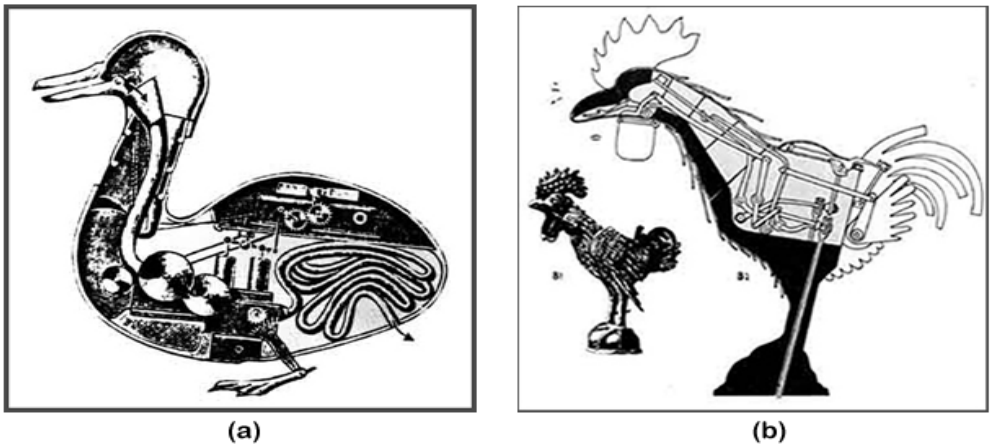


Figura 2.1: (a) Pato de Vaucanson. (b) Gallo de Estrasburgo.

- Leonardo Da Vinci en 1515, construyo para el rey de Francia, un león mecánico, que era capaz de andar y poseía una trampilla en el pecho se abría para mostrar la flor de lis, símbolo de la monarquía francesa.

- En 1739, Jacques de Vaucanson construyó un autómata con forma de pato transparente y compuesto por más de cuatrocientas partes móviles, el cual era capaz de batir las alas, comer e imitar el proceso digestivo de las aves.

- A lo largo del siglo XVIII, Pierre Jaquet-Droz, construyó tres autómatas con forma humana (La Pianista, El Dibujante y El Escritor), cada uno ellos era capaz de realizar un tarea especifica de forma similar a como la realizaría un ser humano.

- Durante la revolución industrial se desarrollaron complejas máquinas capaces de automatizar tareas, anteriormente realizadas por los seres humanos, como la hiladora giratoria construida por Hargreaves en 1770 o el el telar mecánico desarrollada por Cartwright en 1785.

A pesar de los distintos avances realizados a lo largo de los distintos siglos, la robótica y el concepto de robot como los conocemos actualmente, no surgieron hasta principios del siglo XX, debido a los distintos avances tecnológicos que se

fueron sucediendo en distintas áreas, como la mecánica o el surgir de nuevas áreas como la electrónica y la informática, las cuales llevaron a la construcción de los primeros robots modernos.

> *El termino Robot como actualmente lo conocemos apareció por primera vez en la novela Rossum's Universal Robots del autor Karel Capek [4], a partir de la palabra checa Robbota, que significa servidumbre o trabajo forzado. En esta novela se describe a los Robots como máquinas industriales con forma de humanoide que realizaban diversas tareas para los seres humanos. A partir de esta primera definición, el término fue adoptado para definir a aquellas máquinas que son capaces de realizar tareas de manera automática o mediante la teleoperación por parte de un ser humano.*

Aunque realmente el término Robot debería aplicarse sólo a ese conjunto de máquinas que son capaces de realizar un serie de tareas de manera autónoma, es decir sin la intervención directa de un humano, este ha sido y es aplicado para definir a todo tipo de máquinas que son capaces de realizar tareas complejas o peligrosas para un ser humano, bien mediante la utilización de sistemas automáticos, la realización de conjuntos de operaciones de manera secuencial o mediante teleoperación. El proceso de creación de este tipo de máquinas comenzó en los años 40 cuando empezaron a desarrollarse máquinas herramienta las cuales consistían en sistemas teleoperados por humanos o en sistemas que realizaban operaciones muy sencillas de manera secuencial, como por ejemplo el manipulador maestro-esclavo desarrollado por Goertz en 1948 para la manipulación de materiales radiactivos.

En los años 60 comenzaron a construirse los primeros robot industriales que fueron instalados en las plantas de producción de vehículos en cadena, como el robot Unimate que utilizaba control numérico para el gestión de las operaciones. La utilización de este tipo de robots en las cadenas de montaje supuso un aumento de la investigación y el desarrollo de este tipo de dispositivos motivada por la amplia gama de posibilidades que ofrecía. Suscitó el interés de los investigadores

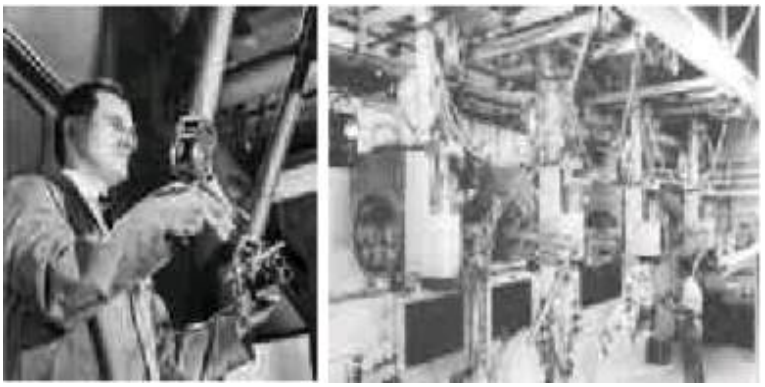

Figura 2.2: Manipuladores Maestro-Esclavo desarrollados por Goertz.

para lograr robots más rápidos, precisos y fáciles de programar, surgiendo los primeros robots controlados por computador a finales de la década de los 60, destacando al robot móvil Shakey [5], el primero de su tipo, desarrollado por el laboratorio de inteligencia artificial de la universidad de Stanford, el cual estaba dotado de diversos tipos de sensores como cámaras de visión y sensores táctiles, y era capaz de realizar operaciones complejas así como reconocimiento de objetos mediante visión artificial básica que le permitían planificar las acciones a realizar.

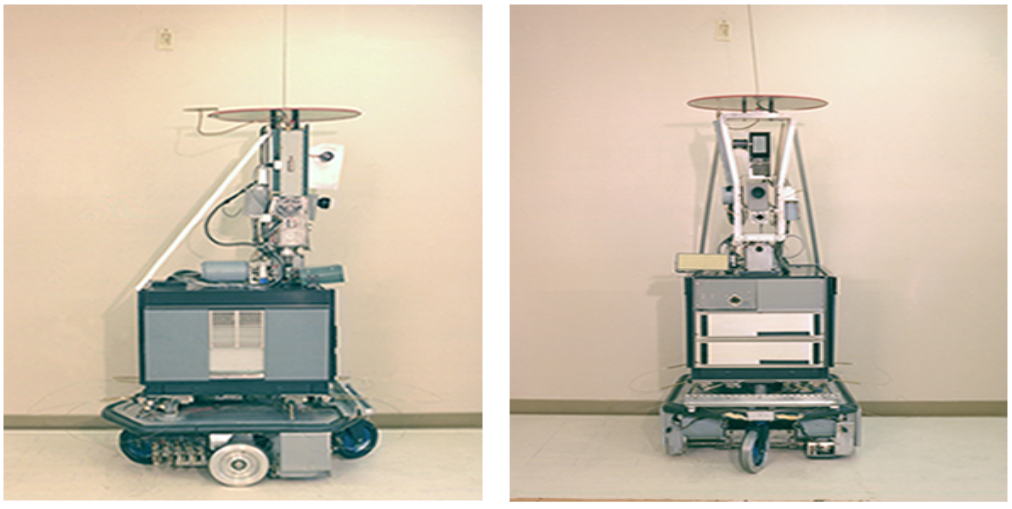

Figura 2.3: Robot Shakey, controlado mediante un sistema deliberativo

A partir de los años 70, la aparición de nuevos tipos de dispositivos mecánicos

y electrónicos más precisos y de menor tamaño, permitieron la creación de agentes físicos, más precisos y complejos, apareciendo una amplia gama de dispositivos en un corto periodo de tiempo.

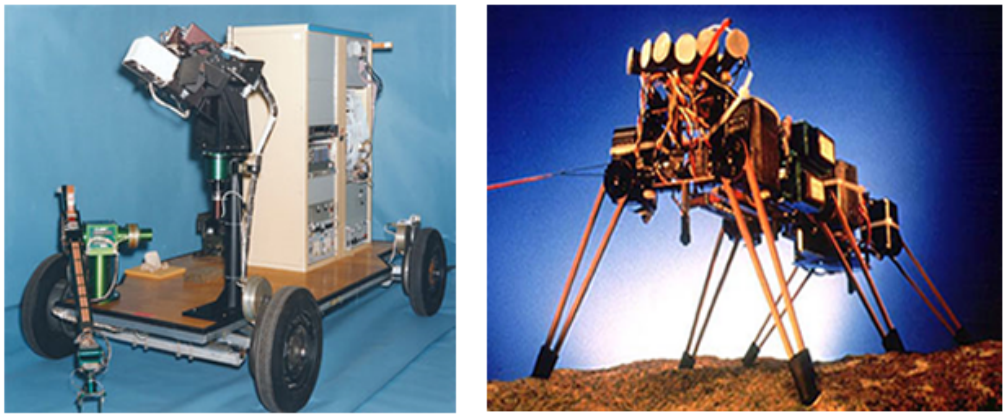


Figura 2.4: (a) Stanford Cart. (b) Robot hexápodo Genghis.

- En el año 1970 se construye en la Universidad de Stanford el Stanford Cart, un robot que es capaz de seguir de forma autónoma las lineas en el suelo.
- En 1978, el robot PUMA (Programmable Universal Machine for Assembly) de la empresa Unimation, es utilizado por la empresa General Motors Co. en sus plantas de producción de automóviles.
- Rodney Brooks del MIT (Massachusetts Institute of Technology) desarrolla en 1989, Genghis, un robot hexápodo con 6 patas articuladas, imitando la anatomía de los insectos.
- En 1992 en Inglaterra, se desarrolla el robot Robodoc (Integrated Surgical Systems. UK), el cual es el primer asistente mecánico en la cirugía de artroplastia de cadera y rodilla.
- LEGO lanza Mindstorms en 1998, dentro de su programa Robotics Invention

System, que permite la construcción y programación de robots de bajo coste de forma sencilla.

- En el año 2000, Honda lanza la última versión de robot antropomorfo, ASIMO (Advanced Step in Innovative Mobility), el cual es capaz de caminar de forma similar a un humano, entablar conversaciones simples y realizar tareas de poca complejidad.

- En 2003, como parte del Programa de Exploración de Marte de la NASA, son enviados los robots autónomos gemelos, Spirit y Opportunity.

- En 2009, la empresa española Pal Robotics presentó el robot Reem-B, un robot humanoide capaz de hablar en el idioma de su interlocutor y localizar su posición en un entorno previamente analizado.

- En 2010, la nasa presentó el robot Robonaut 2 (R2), que es el primer robot autónomo en viajar a la estación espacial.

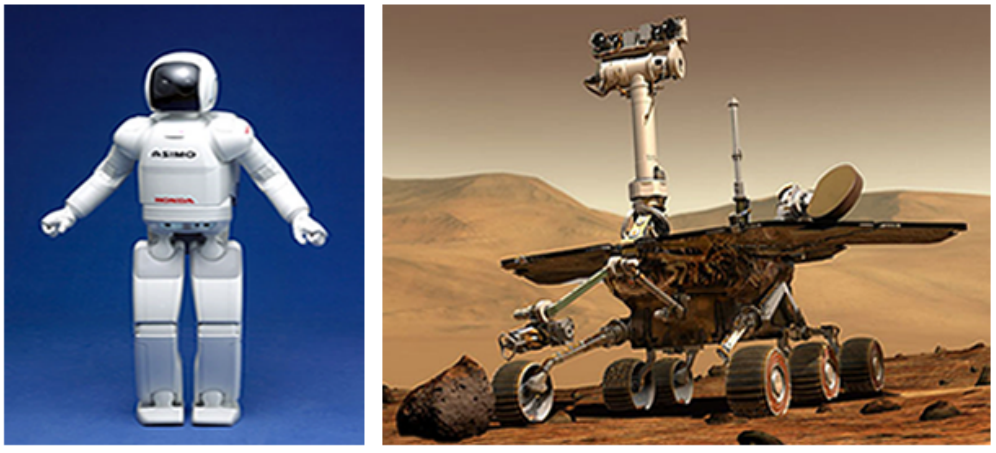


Figura 2.5: (a)Robot ASIMO. (b) Robot Spirit.

En los últimos años la creación de robots ha aumentado considerablemente, no centrándose sólo en sistemas de gestión de tareas industriales o investigación

(espacial, militar, médica, etc), sino en todos los ámbitos de la vida humana, desde juguetes para niños, hasta en sistemas de limpieza automática de viviendas. Teniendo en cuenta este crecimiento tal vez en menos de 20 años todos tengamos un robot en nuestras viviendas de forma similar a como lo imaginó Isaac Asimov a mediados del siglo XX.

## 2.2. Arquitecturas de control

### 2.2.1. Introducción

Las arquitecturas de control, intentan dotar a los agentes físicos de comportamientos inteligentes, pero para poder producir este tipo de comportamiento, el agente o robot debe poder interactuar con el mundo real de forma parcial o total y responder a los distintos cambios que se produzcan en el, con el fin de obtener su objetivo. Para poder interactuar de esta manera los distintos investigadores en el área de la robótica inteligente, llegaron a la conclusión de que todo agente físico debería estar dotado de tres primitivas básicas que le permitieran interactuar con el entorno y conseguir exhibir comportamientos que podrían considerarse inteligentes. Estas son consideradas como las capacidades básicas que un agente físico debe tener:

- Capacidad de percibir: Un agente físico debe ser capaz de obtener información del entorno en el que se encuentra, independientemente de la precisión con la cual esta haya sido obtenida.

- Capacidad de razonar: Un agente físico debe ser capaz de seleccionar y/o deducir que acción (moverse 70 centímetros, girar 90 grados, etc) es más apropiada realizar, teniendo en cuenta la información previamente percibida, la que haya aprendido y/o almacenado anteriormente y la tarea que debe realizar.

- Capacidad de actuar: Un agente físico debe ser capaz de ejecutar la acción

que su sistema de control ha seleccionado previamente de forma precisa.

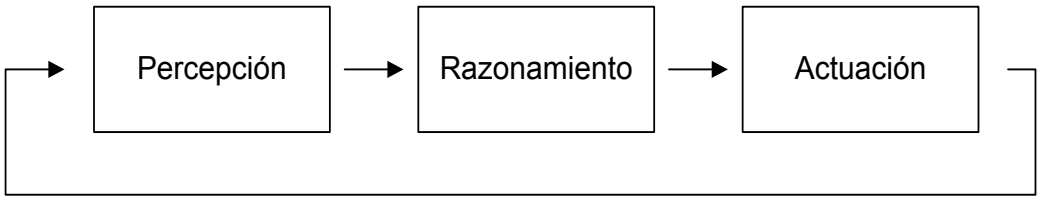


Figura 2.6: Disposición básica de las primitivas de control.

Teniendo en cuenta estas funciones básicas, las cuales pueden ser también denominadas fases del proceso de razonamiento, es posible construir un modelo de razonamiento que permita a un agente físico interactuar en mayor o menor medida con el entorno que le rodea, atendiendo a dos factores:

- La relaciones que se establecen entre las entradas y salidas de las primitivas de control básico. Estas relaciones no sólo se refieren a la distribución de la información, sino al orden y la forma en la cual se ejecutan las primitivas (secuencial, paralelo, basado en eventos, etc).

- Los mecanismos de procesamiento y distribución de la información sensorial a través del sistema de control. En algunos paradigmas la información sensorial es utilizada de forma específica para un única función del agente, en cambio en otros, esta información puede ser utilizada para la construcción de modelos virtuales o relajados del entorno y utilizada de forma general por el sistema de control para tomar decisiones.

La aplicación de estos factores produce la aparición de un amplio conjunto de modelos de razonamiento, los cuales tradicionalmente se dividen en dos grupos. El primero de ellos se basa en la disposición las primitivas básicas o fases de razonamiento de manera vertical, de forma que sólo una de las capas tiene acceso

a los datos obtenidos a través de los sistemas de percepción del robot y por tanto cada una de la capas es responsable de forma individual de la salida y por tanto de la entrada de la siguiente capa. En cambio en el segundo grupo se disponen las capas de razonamiento de manera horizontal, de forma que todas ellas tienen acceso a los datos obtenidos del entorno a través de los sensores del robot y por tanto todas podrán ser responsables de la salida final que será comunicada al robot. En la figura 2.7, se presentan las estructuras básicas de estos dos modelos de razonamiento atendiendo a la disposición de la primitivas básicas de control.

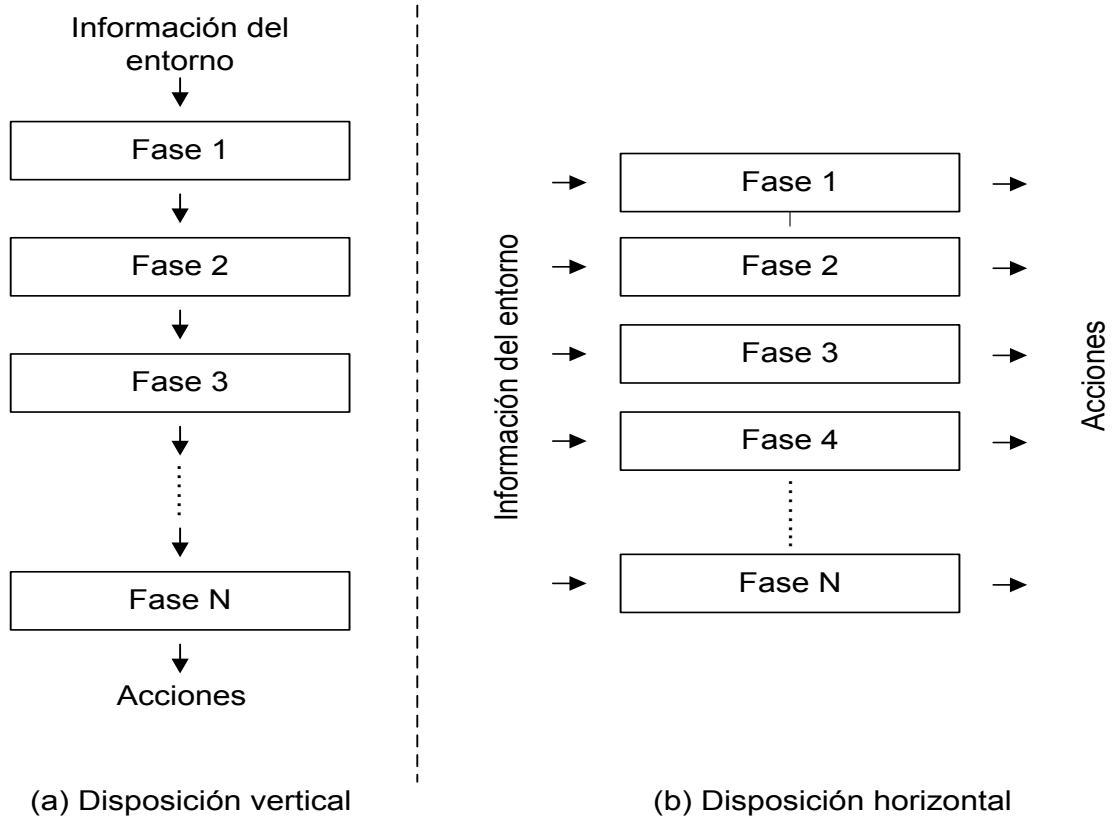


Figura 2.7: Modelos de razonamiento según la disposición de las primitivas básicas.

La utilización de los distintos modelos de razonamiento, ha dado lugar a la aparición de distintas arquitecturas de control, las cuales se clasifican en tres

paradigmas (Deliberativos, Reactivos, Híbridos) que se describen a continuación.

### 2.2.2. Paradigma Deliberativo

La inteligencia artificial puede ser considerada como uno de los precursores de la robótica moderna. Este proceso de evolución conjunta de ambas disciplinas comenzó tras la conferencia de Darthmond en 1956, en la cual los distintos investigadores en el campo de la inteligencia artificial realizaron una diversificación y especialización de los distintos aspectos de esta, como la resolución genérica de problemas, el aprendizaje automático, la visión artificial, etc. A partir de esta especialización comenzaron a producirse los primeros intentos de dotar a los agentes físicos de capacidades inteligentes, mediante la utilización de las técnicas obtenidas fruto de la investigación en las nuevas áreas. Los primeros sistemas de control fueron construidos mediante el Paradigma Deliberativo, el cual está parcialmente basado en los sistemas de resolución general de problemas (GPS, General Problem Solver) creados por Herbert Simon y Allen Newell en 1957 [6].

> De forma general. este paradigma está basado en la premisa de que si un sistema es capaz de utilizar una representación simbólica del entorno que le rodea, entonces podrá generar conductas o acciones que pueden considerarse inteligentes.

Teniendo en cuenta esta premisa, este paradigma se basa en la utilización de las primitivas de control en un orden secuencial, dándole una mayor importancia a la función de razonamiento, en el cual la información obtenida a partir de los sensores es modelada, construyéndose una representación simbólica del estado del entorno y aplicando a continuación técnicas, normalmente de planificación automática, para la obtención de un plan secuencial de acciones, cuya aplicación supone la obtención de los objetivos de la tarea o tareas a realizar. Es necesario destacar que no sólo la información del mundo debe estar modelizada sino también la representación de los elementos que forman parte del mundo, así como todas

aquellas operaciones que pueden ser aplicadas en el entorno por el agente físico. En la figura 2.8, se presenta la estructura básica de la arquitectura deliberativa.

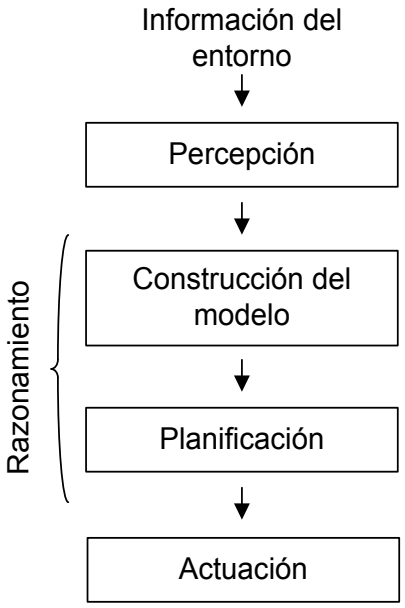


Figura 2.8: Modelo de Razonamiento del paradigma deliberativo.

A pesar de los buenos resultados que se obtuvieron en entornos reducidos y muy específicos, este tipo de arquitectura posee un problema de escalabilidad en entornos muy volátiles, en parte debido a los algoritmos que son utilizados para la resolución de los problemas de planificación, en los cuales se asume la propiedad de **racionalidad calculada** definida por Russell en 1993 [7], y que consiste en lo siguiente:

> Dado un conjunto de información obtenida del mundo en un instante de tiempo **t**, la acción obtenida tras el proceso de planificación realizado utilizando la información previamente obtenida, debe ser aplicada en el instante de tiempo **t** en el cual comenzó el proceso de razonamiento (proceso de planificación) para poder ser óptima.

Esto supone que no puede producirse ningún cambio en el entorno durante el proceso de deliberación. Esta propiedad no se cumple en entornos muy volátiles

o donde el tiempo de deliberación es muy elevado, lo cual produce que el conjunto de acciones obtenido no tenga en cuenta nuevos aspectos del entorno.

Teniendo en cuenta que fue el primer paradigma de control aplicado al control de robots, supuso grandes cambios en la forma de construir este tipo de dispositivos, pero también demostró que su utilización suponía la aparición de graves problemas:

- Como ya describimos anteriormente este paradigma necesita conocer toda la información del entorno, con el fin de realizar una representación simbólica del entorno y aplicar un algoritmo de planificación, para generar el conjunto de acciones necesarios para producir un comportamiento inteligentes que resuelva el problema.

- El conjunto de recursos que necesita el algoritmo de planificación crecen según aumenta la complejidad del entorno o el número de metas, de forma que la escalabilidad de este tipo de técnicas es bastante reducida.

- Algunas de las acciones seleccionadas pueden ser obsoletas con respecto al entorno, o no poder ser ejecutadas. Por ejemplo, imaginemos que existe un objeto móvil que el robot debe evitar, si el proceso de planificación consume un tiempo muy elevado el objeto móvil se encontrará en otra posición, probablemente muy alejada de la que fue detectada inicialmente y tal vez el robot podría chocar con él al realizar algunas de las acciones planificadas.

#### 2.2.2.1. Arquitectura Shakey

Shakey [5] no es realmente una arquitectura de control, es el nombre con el cual se conoce al primer robot que utilizó una arquitectura de control de tipo deliberativo con el fin de producir comportamientos inteligentes. La arquitectura desarrollada para este robot es similar a la definida en la descripción teórica de este paradigma. Se encuentra constituida por tres capas, dispuestas de manera jerárquica de forma que las capas superior generan las entradas de las capas in-

feriores. En la figura 2.10, se presenta un diagrama de las distintas capas que formaban la arquitectura de control desarrollada para el robot Shakey.

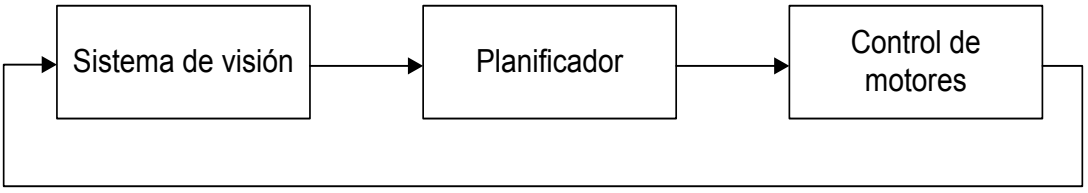


Figura 2.9: Arquitectura de control desarrollada para el robot Shakey.

- La capa de percepción, se encarga de obtener la información de los distintos sensores, en el caso del robot Shakey, el sistema de sensores se limitaba a una cámara, que obtenía información del entorno y procesaba esta información enviándola a continuación a la siguiente capa.

- La capa de razonamiento, se encarga de la transformación de la información obtenida, de forma que esta pueda ser añadida en el modelo del entorno que posee el sistema, el cual será utilizado posteriormente para encontrar un plan de acciones a realizar mediante la aplicación de un planificador.

- La capa de actuación, se encarga de comunicar las distintas acciones a realizar a los actuadores del robot, en el caso de este robot .

El sistema utilizado en esta arquitectura puede definirse como un proceso, el cual se repite n veces en el tiempo, formado por tres acciones observar, pensar o razonar y actuar. Donde la acción de razonamiento era la más compleja y la que mayor coste computacional tenía. El proceso de razonamiento, mediante el cual se obtienen las ordenes a realizar por parte del robot, es realiza mediante la utilización de un planificador denominado Strips, desarrollado por Richard Fikes and Nils Nilsson en 1971 [8].

#### 2.2.2.2. Arquitectura JPL

A principios de la década de los noventa, se presento la arquitectura JPL (Jet Propulsion Laboratory) [9], este intentaba ofrecer un sistema de navegación automática para vehículos de exploración planetaria. El sistema está formado por dos tipos de dispositivos, un vehículo explorador terrestre y un satélite, normalmente de tipo geoestacionario, que obtiene información detallada del entorno en el que se encuentra el vehículo. La estructura de los distintos componentes de la arquitectura, presentada en la figura 2.11, se encuentra dividida en cuatro módulos:

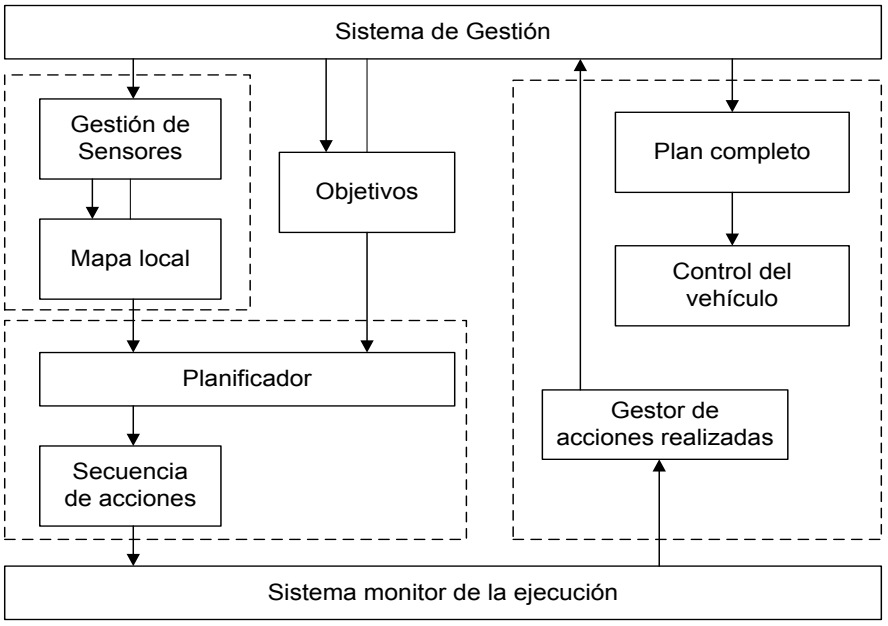


Figura 2.10: Arquitectura JPL (Jet Propulsion Laboratory).

- Módulo de percepción: Este módulo se encarga de la obtención de información del entorno a través de los distintos vehículos que forman parte del sistema. La información recogida es utilizada para la creación de mapas de tipo local y global del entorno, los cuales son utilizados en el proceso de planificación.

- Módulo de planificación: Este módulo se encarga de la generación de un plan de acciones libre de colisiones, mediante la utilización del mapa local generado por el módulo de percepción. La longitud de los caminos generados es de una longitud aproximada de 10 metros.

- Módulo de monitorización: Realiza una simulación del plan generado para el robot, realizando pequeñas modificaciones y teniendo en cuenta posibles fallos que puedan producirse durante el proceso de ejecución, este sistema incluye un conjunto de maniobras predefinidas para evitar situaciones complejas no tenidas en cuenta durante el proceso de planificación de la ruta.

- Módulo de ejecución: Es el sistema central que rige el funcionamiento de la arquitectura y se encarga de llevar a cabo el plan de ejecución generado por el módulo de planificación.

Este arquitectura, no es totalmente deliberativa, debido a que posee un conjunto de maniobras o comportamiento iniciales, que pueden ser utilizados en ciertas situaciones para evitar problemas y situaciones no tenidas inicialmente en cuenta por el módulo de planificación, por lo que algunos investigadores consideran que posee algunos componentes de las arquitecturas reactivas.

#### 2.2.2.3. Conclusiones

Las arquitecturas de tipo deliberativo, ofrecen un proceso que es capaz de producir un plan de acciones que permite a un agente físico o robot resolver un problema, pero conlleva normalmente un gran coste computacional, debido a los distintos algoritmos o sistemas que son utilizados para planificar tareas o trayectorias, que junto con el problema de la **racionalidad calculada**, limitan mucho la utilización de las arquitecturas deliberativas como sistema de razonamiento en entorno volátiles o de una elevada complejidad.

### 2.2.3. Paradigma Reactivo

Los distintos problemas que surgieron de la aplicación del paradigma deliberativo debido a su falta de adaptabilidad y a los costes computacionales que suponía la obtención de los planes de acciones en entornos reactivos produjeron la aparición, a finales de los años 80, de las primeras críticas a la utilización de sistemas basados en representaciones simbólicas para construir sistemas de control para agentes físicos.

Uno de los principales artífices de estás críticas, fue Rodney Brooks, el cual afirmó que no era posible descomponer la inteligencia humana en un conjunto de áreas o tareas básicas, debido a su elevada complejidad y al limitado conocimiento que se posee de su funcionamiento [10]. Por lo tanto, según sus teorías, no es eficiente utilizar una representación simbólica como modelo para definir el mundo real, debido a que para su construcción un amplio número de aspectos de este son simplificados llevándonos a un modelo que refleja de forma muy limitada el mundo. Por lo tanto la mejor solución es utilizar el propio mundo real como modelo en vez de una abstracción de este, independientemente de la forma de abstracción que sea empleada para construirlo.

Basándose en estos postulados, Brooks desarrollo una teoría que afirma que los comportamientos inteligentes de un robot surgen de su interacción con el mundo, que producen la aplicación de distintos comportamientos sencillos (instintos, reacciones, etc) que en conjunto son capaces de producir comportamientos más complejos, los cuales pueden considerarse como comportamientos o acciones inteligentes, este tipo de comportamientos se corresponden con un proceso de acción - reacción, o aplicado a los seres vivos como un proceso de estímulo - respuesta. Esta teoría dio lugar al **paradigma reactivo** el cual tiene su máxima expresión en la arquitectura subsunción [11] [12] desarrollada por Brooks con el fin de aplicar sus teorías. En la figura 2.12, se presenta un diagrama que presenta la estructura básica de un modelo de razonamiento basado en el paradigma reactivo.

La estructura del paradigma reactivo se basa en la **separación de las fases**

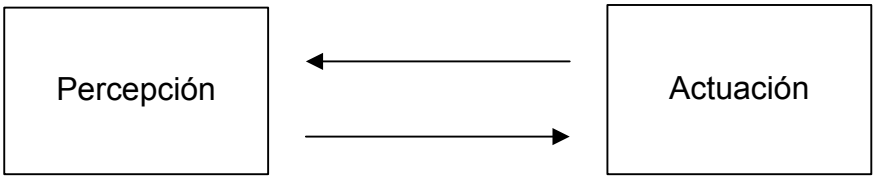


Figura 2.11: Modelo de razonamiento del paradigma reactivo

**de razonamiento en pequeñas actividades**, las cuales se encuentran dispuestas en paralelo como se describió en el modelo de razonamiento horizontal de la figura 2.7. Cada una de estas actividades recibe información del entorno a través de los sistemas sensoriales del robot y genera una salida la cual puede ser combinada o no con las otras salidas con el fin de generar un conjunto de acciones que deberá realizar el robot. Las principales ventajas que aporta esta nuevo paradigma respecto a su predecesor, el paradigma deliberativo, son tres:

- **Reacción parcialmente automática** a los cambios en el entorno, lo cual ya supone un cambio frente al paradigma deliberativo, el cual se encuentra sujeto a la propiedad de racionalidad calculada.
- **Realización de múltiples acciones en paralelo**, debido a la disposición paralela de las tareas o comportamientos.
- Posibilidad de **resolución de objetivos de forma paralela**, lo cual es totalmente contrario a los sistemas de resolución empleados en el paradigma deliberativo, en el cual todas la metas o un amplio número de ellas intentan ser resueltas de forma conjunta mediante el proceso de razonamiento (proceso de planificación), lo cual aumenta el coste computacional del proceso de razonamiento según aumenta el número de metas.

#### 2.2.3.1. Subsunción

La arquitectura más conocida de tipo reactivo fue la desarrollada por Brooks, denominada **Subsunción**, la cual consiste en la aplicación de un conjunto de tareas, dispuesta de manera paralela en capas, cada una de ellas construida mediante

un máquina de estados finita, las cuales interactúan entre sí mediante dos mecanismos:

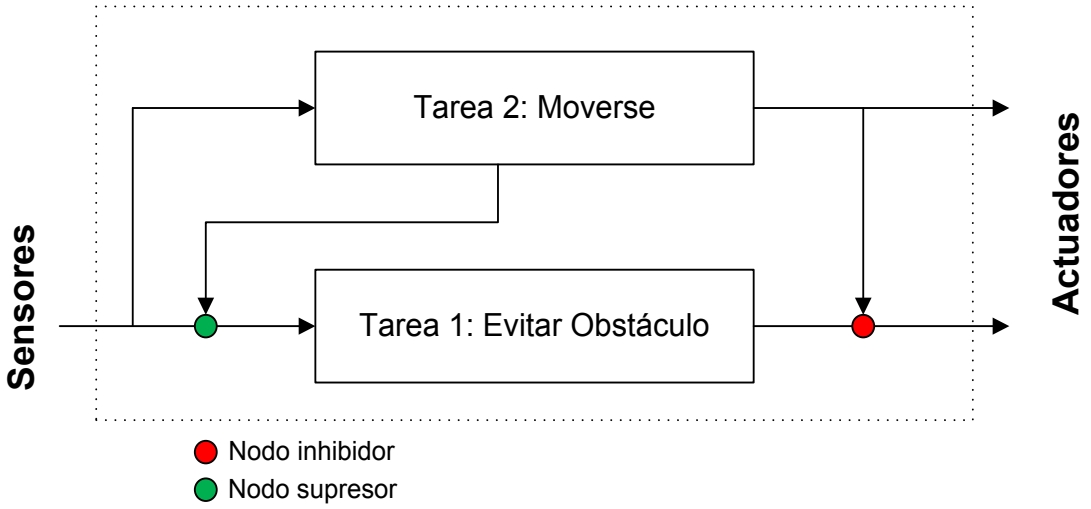


Figura 2.12: Ejemplo de un controlador basado en subsunción.

- Supresión: Una tarea es capaz de suprimir las entradas que llegan a otra tarea, sustituyéndola por otra. Normalmente esta nueva entrada se corresponde con la salida generada por la tarea que produce la supresión.
- Inhibición: Una tarea es capaz de inhibir la salida de otra tarea, sustituyéndola por otra. Normalmente esta nueva salida se corresponde con la salida generada por la tarea que produce la inhibición.

El proceso de interacción se produce de forma jerárquica, es decir, sólo las capas superiores pueden aplicar supresión o inhibición sobre capas inferiores, debido a que la arquitectura subsunción para un robot debe construirse mediante el desarrollo de comportamientos muy sencillos a comportamientos más complejos de forma que aquellos comportamientos más complejos deben aprovecharse de los sencillos para poder producirse. Esto supone no sólo la aplicación de una arquitectura de control específico, sino también la aplicación de una metodología de programación conocida como **bottom-up**, donde la programación de un sistema

se realiza desde las funciones más específicas hasta las más generales, en este caso, de las más sencillas a las más complejas.

#### 2.2.3.2. Campos potenciales

Los campos potenciales es una arquitectura reactiva similar a subsunción, con la diferencia de que en vez de capas de comportamientos, se construyen capas de gestión de campos potenciales, los cuales interactuaran entre si de forma muy sencilla generando comportamientos más complejos que pueden considerarse inteligentes.

Su funcionamiento se basa en el comportamiento de las partículas subatómicas, las cuales se repelen cuando poseen cargas eléctricas similares. Este comportamiento se transfiere a los distintos objetos del entorno de forma que los robots y los objetivos o metas poseerán campos potenciales similares por lo cual se atraerán entre sí, y los obstáculos campos potenciales opuestos de forma que el agente móvil los repelerá alejándose de ellos.

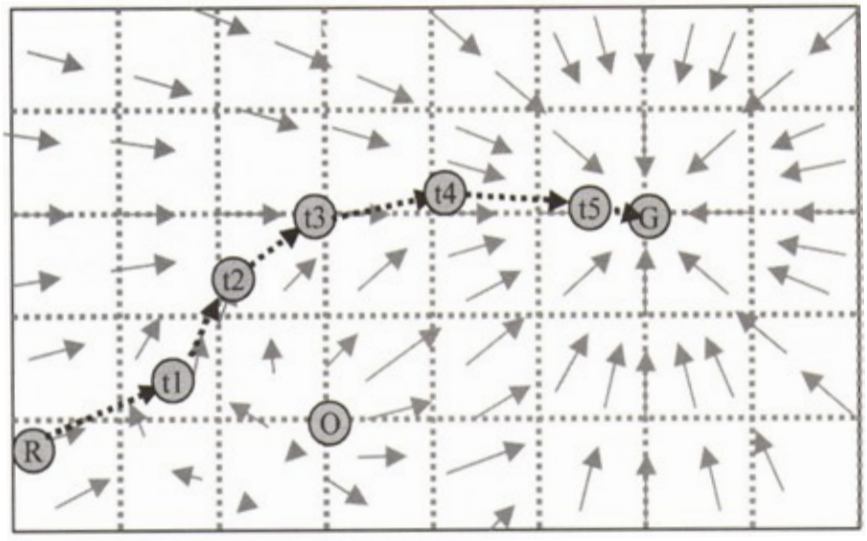


Figura 2.13: Ejemplo de aplicación de campos potenciales.

El funcionamiento de esta técnica se basa en la realización de una evaluación de la posición del agente en todos los puntos del entorno conocido teniendo

el cuenta la influencia que ejercen los distintos objetos del entorno (meta, obstáculos). Para cada punto, que denominaremos p, se realiza el calculo del potencial mediante la función de potencial U, definida sobre el espacio libre, y compuesta por la unión del potencial atractivo $U_a$ (generado por el destino) y el potencial repulsivo $U_b$ (generado por los distintos obstáculos que se encuentran próximos al agente físico), es decir:

$$U(p) = U_a(p) + U_b(p) \tag{2.1}$$

Debido a este potencial, sobre el agente físico actúa una fuerza artificial F en el punto p, que se calcularía mediante la siguiente formula:

$$F(p) = -\nabla U(p) \tag{2.2}$$

Al igual que la función potencial, la fuerza sobre el agente físico se descompone en una fuerza atractiva que le dirige hacia el destino y una repulsiva que le aleja de los obstáculos del entorno, de la siguiente manera:

$$F(p) = F_a(p) + F_b(p) \tag{2.3}$$

Los valores de potencial van variando según la posición del agente físico, de forma que el campo potencial de atracción disminuirá según el agente se acerque a la meta, y el potencial de repulsión es un valor 0 hasta cierta distancia al objeto y tenderá a infinito según el agente se vaya acercando al obstáculo.

Teniendo en cuenta la descripción anterior, se si desea construir un algoritmo que utilice las técnicas de campos potenciales para guiar un robot, se debería construir un proceso formando por los siguientes pasos:

1. Calcular el potencial en un punto U(p) según la información aportada por los distintos sensores del robot.

2. Calcular la fuerza F(p) sobre el agente físico en ese punto.

3. Calcular el movimiento del robot teniendo en cuenta los parámetros previamente calculados (Dirección y sentido del movimiento, teniendo en cuenta el valor de la función F en el punto p).

Actualmente existen múltiples técnicas para calcular las fuerzas potenciales del entorno. Una de ellas consiste en calcular las fuerzas en función de la distancia euclídea, de manera que cuando el robot móvil se aproxime a dicho punto, disminuya su influencia; mientras que la fuerza repulsiva sólo influya cuando el móvil se acerque a un objeto a partir de una determinada distancia y se vaya incrementando mediante una función que invierta el valor de la distancia euclídea entre el obstáculo y el agente.

Este tipo de técnica es muy útil en entornos en los cuales sólo se trabaja con información parcial del entorno, por lo que tiene gran utilidad en entorno dinámicos y muy volátiles. Aunque tiene un defecto, ya que se pueden producir zonas del entorno en las cuales se produzcan anulaciones de los campos atractores y repulsores dejando al robot ciego en ese punto, o producir bucles de los cuales el robot no puede salir y finalizar su objetivo.

#### 2.2.3.3. Conclusiones

A pesar de sus aparentes ventajas y de los buenos resultados obtenidos en una amplio número de experimentos, la aplicación de este tipo de arquitectura en entorno reales supone la aparición de ciertos problemas:

- En primer lugar se asume que con la información local que es capaz de obtener el robot mediante su sistema sensorial puede realizar de forma correcta su tarea y producir comportamientos inteligentes. Pero esto no es del todo cierto, ya que la información obtenido puede no ser suficiente para la realización de las acciones necesarias.

- Existen muchos problemas a la hora de aprender del entorno o durante el proceso de interacción, debido a la simplicidad de los comportamientos aplicados en la estructura.

- La aparición de comportamientos complejos se produce, pero no siempre aparecen los comportamientos esperados, incluso en muchos casos algunos de estos comportamientos emergentes son complejos de gestionar o incluso de depurar en el caso de que fueran comportamientos erróneos.

- La generación de comportamientos realmente inteligentes es muy complejo y supone la utilización de un número elevado de tareas, lo cual aumenta la complejidad del sistema de control del robot, principalmente debido a las relaciones que deben establecerse entre las distintas capas.

### 2.2.4. Paradigma Híbrido

A principio de la década de los 90, los investigadores en el campo de la robótica inteligente llegaron a la conclusión de que los paradigmas diseñados hasta el momento no se adaptaban bien a entornos reales, debido a que cada uno de ellos se adaptaba a un pequeño conjunto de problemas, pero resultaban inadecuados para un número mayor de ellos. Esto supuso que un gran número de investigadores decidiera aunar las ventajas de los dos paradigmas existentes en uno nuevo que fue denominada **paradigma híbrido**.

Este nuevo paradigma combina las ventajas de los dos anteriores, tomando del paradigma reactivo los sistemas de procesamiento de estímulos a nivel local sin necesidad de aplicar un sistema de deliberación complejo, permitiendo al sistema ofrecer respuesta en tiempo real a ciertas acciones básicas, pero también componentes del paradigma deliberativo de forma que pueden construirse estrategias de comportamiento a medio y largo plazo que permitan al sistema obtener sus objetivos o incluso crear mecanismos de interacción con otros agentes físicos independientes.

Debido a la naturaleza híbrida de este paradigma no se puede realizar una clasificación del tipo de modelo de razonamiento que utilizará, así como del número de capas o las relaciones que existen entre ellas. A pesar de las grandes diferencias que existen entre las distintas arquitecturas híbridas, estas normalmente están

construidas mediante tres capas:

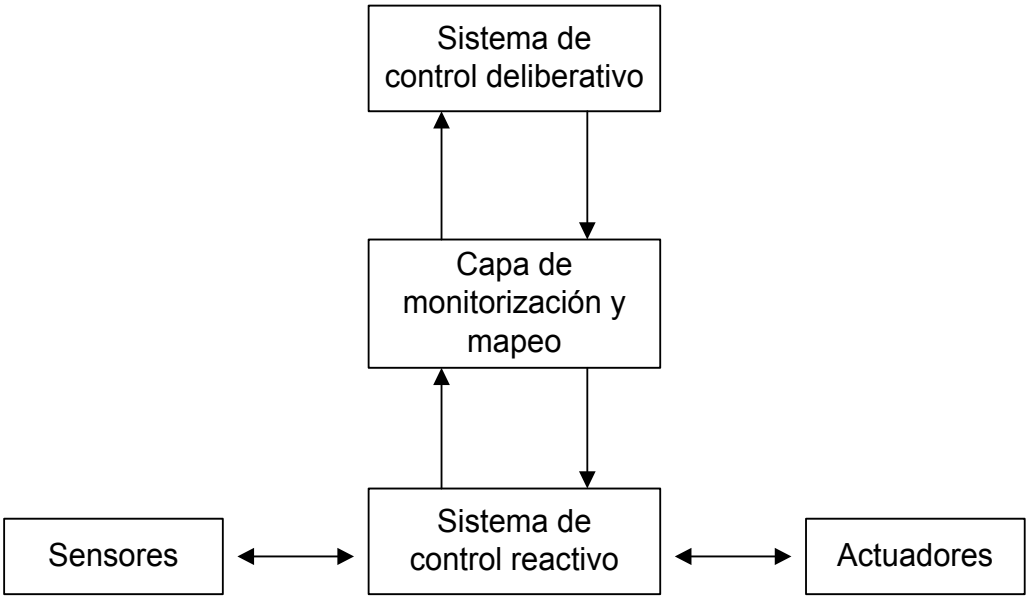


Figura 2.14: Modelo de Razonamiento del paradigma híbrido.

- En nivel inferior de este tipo de paradigma, es considerado como el sistema reactivo, ya que se ocupa de la obtención de la información de los sensores y la aplicación de comportamientos reactivos teniendo en cuenta la información obtenida.
- El segundo nivel, consiste en una capa de interacción entre el sistema de control reactivo y el sistema de control deliberativo, en la cual normalmente se puede almacenar información de control para la interacción con otros agentes físicos, conocimiento adquirido del entorno o incluso conocimiento previo.
- Este tercer nivel, se corresponde con el sistema deliberativo del robot, en el se produce la generación de los distintos comportamiento complejos aplicables a largo plazo.

Hay que tener en cuenta que la especialización por capas descrita debe ser tomada como un patrón de diseño básico, debido a que existen un amplio número

de arquitecturas híbridas las cuales poseen menos niveles, que utilizan tres capas pero con distintas funcionalidades a las descritas en este documento [13], o que incluyen un número mayor de niveles. Normalmente estos nuevos niveles surgen de la descomposición de los niveles aquí descritos en dos o más debido a la complejidad con la que se ha dotado a la capa, o bien por comodidad del diseñador de la arquitectura, o actualmente debido a la inclusión de capas que tratan de producir comportamiento similares a las emociones o relaciones entre distintos robots o agentes [14].

#### 2.2.4.1. Arquitectura Atlantis

La arquitectura Atlantis, fue una propuesta desarrollada a principios de la década de los 90, en el Instituto Tecnológico de California [15]. Fue definida como una arquitectura heterogénea y asíncrona basada en un modelo de estado-acciones [16]. Aunque el modelo utilizado en esta arquitectura difiere del modelo de estados-acciones clásico, debido a que las acciones son considerados como **acciones continuas** y el coste de aplicación de cada una, es decir el proceso de computación que conlleva la ejecución de una acción, es despreciable.

En Atlantis, este tiempo de computación de una acción es denominado actividad y las operaciones que inician y finalizan las actividades son denominadas decisiones. Por lo tanto una decisión siempre inicializará una actividad, la cual a su vez puede iniciar otras actividades, construyendo de esta manera una estructura jerárquica de tareas, que produce que las actividades más complejas se dividan en actividades más sencillas, llegándose a generar actividades que no pueden iniciar otras actividades. Estas últimas son las denominadas primitivas, debido a que su coste computacional es despreciable. Para conseguir esta descomposición la estructura de Atlantis está dividida en tres componentes dispuestos de forma vertical, como se puede observar en la figura 2.16.

- Controlador, se corresponde con el nivel más bajo del sistema, lo que produce que gestione los distintos sensores y actuadores y por tanto es el encargado de gestionar las actividades primitivas o reactivas. Cada una de estas funciones

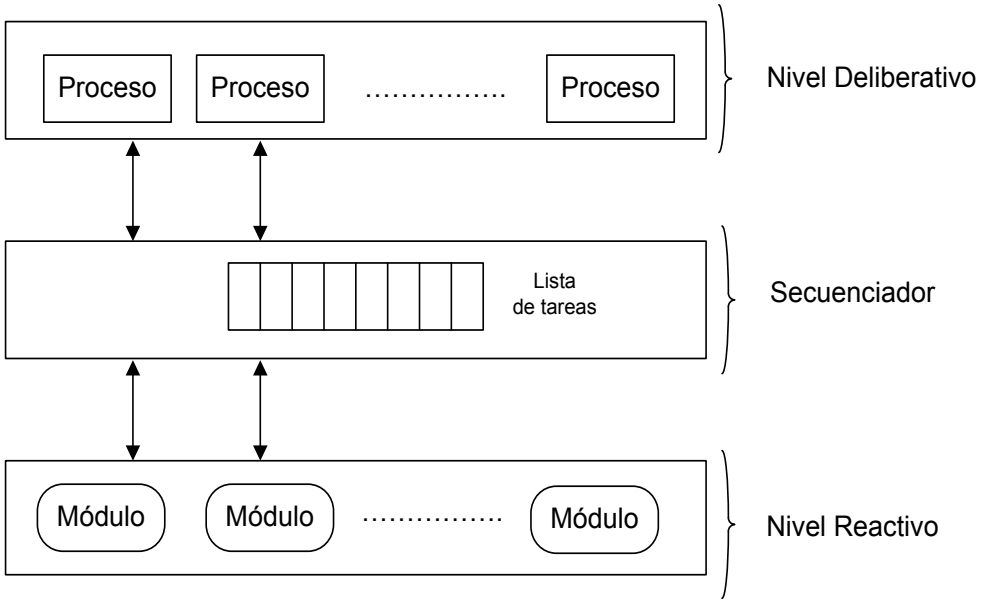


Figura 2.15: Modelo de razonamiento de la arquitectura Atlantis.

primitivas está formada por uno o más módulos, los cuales son desarrollados en un lenguaje nativo de la arquitectura denominado ALFA, que permite disminuir el coste computacional de las primitivas.

- El secuenciador, es el organizador de actividades, ya que se encarga de controlar la secuencia de actividades. El funcionamiento de este modulo es muy complejo, ya que debe ordenar de forma automática las distintas acciones a ejecutar y tener un control de las ya ejecutadas, que pueden influir en el orden de las acciones que vayan a ejecutarse. Una secuenciación incorrecta puede no producir ningún error en las actividades del robot o puede producir un fallo crítico, por lo tanto las funciones de este sistema son críticas.

- La última capa, la encargada del razonamiento, se corresponde con el nivel deliberativo, encargándose de las tareas computaciones como la planificación, o la gestión y/o modelización del mundo. Se encuentra formando por un conjunto de programas codificados en LISP, los cuales representan distintos algoritmos de búsqueda heurística o planificación.

En esta arquitectura se pueden apreciar con facilidad la ventajas de las arquitecturas híbridas, entre las cuales se observa una diferenciación clara entre los comportamientos simples que puede producir un robot y los comportamientos complejos, aquellos que podemos considerar inteligentes, los cuales se obtiene mediante la descomposición de las actividades generadas por los algoritmos de inteligencia artificial. En definitiva la capa de secuenciación y la capa de control, son un complejo interfaz que permite a los algoritmos de planificación o búsqueda heurística interactuar con los sensores y actuadores del robot.

#### 2.2.4.2. Arquitectura AuRa

AuRa (Autonomous Robot Architecture), fue desarrollada por Ronald Arkin en 1989 [17]. Su funcionamiento se basa en la teoría de esquemas y se encuentra fuertemente influenciada por varias teorías médicas referente a la psicología y a la neurofisiología. Según Rumelhart [18], un **esquema** es una estructura de datos, utilizada para representar conceptos genéricos almacenados en la memoria humana, por tanto la **teoría general de los esquemas** se encarga de definir la forma mediante la cual se representa el conocimiento y de cómo este es utilizado.

La estructura de esta arquitectura está compuesta de dos módulos, uno de bajo nivel de tipo reactivo y otro de alto nivel de tipo deliberativo. En la figura 2.17 se puede observar de forma más detallada los elementos de cada una de las capas.

La primera de la capas, el nivel reactivo, funciona mediante la utilización de esquemas o comportamientos sencillos, los cuales pueden combinarse para construir comportamiento más complejos. La complejidad de estos comportamientos dependerá del número de esquemas que sean utilizados para producirlos y de su complejidad. El nivel reactivo está formando a su vez por tres componentes:

- Sistema de percepción: Esta formando por los distintos esquemas de percepción que posee el sistema, que permiten la recogida de la información a partir de los distintos sensores y la transmiten al sistema cartográfico y al

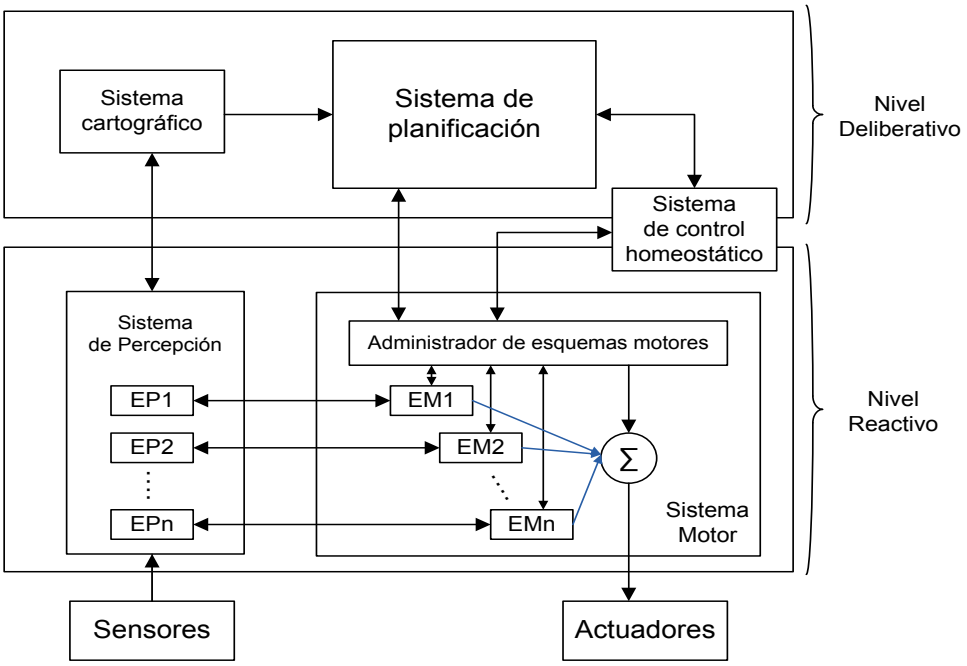


Figura 2.16: Modelo de razonamiento de la arquitectura AuRa.

sistema motor. La información recibida a través de los sensores puede sufrir transformaciones por parte de los distintos esquemas de percepción.

- Sistema motor: Esta formando por los distintos esquemas motores, que permiten la generación de los comportamientos que realizará el robot. Para la activación de los distintos esquemas motores, así como la coordinación que existirá entre ellos, el sistema obtendrá información del sistema de percepción y del sistema de planificación del nivel deliberativo.

- Sistema homeostático: Este sistema puede considerarse uno de los más importantes de la arquitectura, ya que es el encargado de mantener un estado de equilibrio entre los distintos elementos de la arquitectura. Por lo tanto tiene control sobre ambos niveles pudiendo manipular los comportamientos del robot, así como el estado interno de este.

El nivel deliberativo, el cual se encarga de gestionar las acciones a largo plazo, está formado por dos componentes:

- Sistema cartográfico: Es el encargado de gestionar el razonamiento espacial del robot. Está compuesto de dos sistemas de almacenamiento de información, uno para la construcción de mapas globales, que puede considerarse como una memoria a largo plazo, y otra que almacena la información obtenida mediante los sensores en las últimas lecturas, algo simular a un *buffer* que es modificado tras un tiempo o tras la obtención de una cantidad de información determinada.

- Sistema de planificación: Es el encargado de realizar la navegación deliberativa, es decir es el responsable de planificar las acciones que debe realizar el robot, esta formado por tres módulos:

  - Módulo de misión: Se encarga de recoger y preparar los datos necesarios para que se produzca el comportamiento esperado.

  - Módulo de navegación: Se encarga de generar la trayectoria que utilizará el robot, teniendo en cuenta los datos seleccionados por el módulo de misión.

  - Módulo de pilotaje: Utiliza la trayectoria planificada y selecciona los distintos comportamientos que deberán ser realizados. Este sistema realiza la activación de los esquemas seleccionados para producir los comportamientos planificados.

La gran ventaja de esta propuesta es la modularidad de su arquitectura que permite fácilmente modificar algunos de sus sistemas o módulos, siempre que se respeten las interacciones entre ellos, esto le permite ser utilizado en cualquier robot realizando un número reducido de cambios. Además es capaz de producir comportamientos complejos muy adaptativos al entorno, pero en parte debido a que el uso de inteligencia artificial es muy reducido.

### 2.2.5. Conclusiones

A nivel general las arquitecturas híbridas, no son más que la definición de un interfaz que permita utilizar de forma conjunta, con algunas modificaciones, una arquitectura reactiva y una deliberativa, utilizando cada una de ellas para aquellas situaciones en las que mejor se comportan. Debido a esto no existen un gran número de diferencias significativas entre los dos ejemplos descritos en este documento, sino más bien en como se interconectan las capas deliberativa y reactiva, o el grado de actividad de cada una de ellas. Es decir, la capa deliberativa normalmente es utilizada para construir comportamientos complejos o incluso estrategias de control orientadas a resolver problemas complejos de tipo global. En cambio, la capa reactiva es utilizada para resolver situaciones que se producen al intentar ejecutar los comportamientos complejos o las estrategias de control, y que no suponen el incumplimiento de las condiciones mediante las cuales estas fueron definidos, es decir su función es la de resolver problemas de tipo local debido a variaciones en el entorno.

## 2.3. Planificación automática

El proceso de razonamiento humano es una tarea muy compleja; debido no sólo por la complejidad de los distintos procesos que gobiernan nuestros sistemas de razonamiento, sino por el desconocimiento que tenemos del verdadero funcionamiento de nuestro cerebro. A pesar del desconocimiento sobre el funcionamiento de los sistemas de razonamiento humanos, muchos investigadores han teorizado que las técnicas de razonamiento que utilizamos probablemente se basen en la construcción de **planes de acciones**, muy relajados, que luego son completados con procesos reactivos para conseguir obtener nuestro objetivo. Es decir imaginemos que quiero desplazarme de la universidad a mi casa, de forma directa o incluso indirecta; mi sistema de razonamiento construirá un plan de acciones que podría estar formado por ejemplo (ir de la universidad al metro, subir al metro,

bajar del metro, ir del metro a casa), este plan es muy relajado ya que no tiene en cuenta muchas acciones que podrían producirse, como si me sentaré o no en el vagón, si tendré que evitar a un conjunto de transeúntes en mi recorrido, etc. Pero esta formando por un conjunto de acciones, tal vez previamente aprendidas, que producen la consecución del objetivo que es llegar a casa. Una buena opción para simular este proceso de razonamiento es imitarlo, lo cual se lleva realizando desde hace más de 50 años, mediante los sistemas de resolución general de problemas que han evolucionado hasta los planificadores modernos y que han sido utilizados en algunos casos para intentar producir comportamientos inteligentes en agentes físicos.

### 2.3.1. Introducción

La planificación automática es una rama de la inteligencia artificial que tiene su origen en el primer sistema de resolución general de problemas (GPS, General Problem Solver) creado por Herbert Simon y Allen Newell en 1957 [6], este sistema tenía como objetivo resolver problemas de tipo general, que debían ser representados mediante un lenguaje de alto nivel, por medio de un único algoritmo.

Durante las siguientes décadas, los investigadores en el área de la planificación automática fueron refinando sus ideas y teorías llegando a producir un amplio número de sistemas planificación basados en distintas técnicas, pero con una serie de características comunes, la mayor parte de ellas heredadas de los sistemas de resolución general de problemas.

- La primera de ellas consiste en definir el **modelo conceptual**, es decir, las características de los problemas que van a ser resueltos por el sistema y del mundo en el cual van a ser resueltos. Esto supone que el modelo elegido deberá contener:

    1. La información que permita describir la estructura y el funcionamiento del mundo, es decir las distintas acciones que pueden aplicarse para producir modificaciones.

2. Una descripción completa o parcial del estado inicial en el cual comenzará a resolverse el problema.
3. Una descripción del conjunto de objetivos que deben ser conseguidos para resolver el problema.

- En segundo lugar, será necesario un **lenguaje de representación de alto nivel** que permita describir de forma general cualquier problema a resolver por parte del planificador. Los planificadores son sistemas de resolución general de problemas, lo que implica que para comportarse de esta manera deben utilizar un lenguaje que les permita describir cualquier problema, que pueda ser resuelto mediante la utilización de un algoritmo general de búsqueda, normalmente este lenguaje se utiliza para representar dos tipos de información, el primero se corresponde con el dominio de la planificación, el cual está constituido por la descripción de los distintos objetos que pueden aparecer en el dominio, sus características y las distintas operaciones que son capaces de modificar el estado del entorno. El segundo de ellos se corresponde con el problema a resolver y en él se describe el estado inicial en el cual se encuentra el mundo y los distintos objetivos o metas que deben conseguirse para resolver el problema. Esta separación entre dominio y problema no se realiza de forma arbitrara sino con el fin de poder definir un dominio específico de forma general y poder utilizarlo para resolver cualquier problema que corresponda a ese dominio.

- Por último utilizar un **sistema de resolución general** o **planificador**. Este debe ser capaz de resolver los problemas que ha sido representados mediante el lenguaje de representación seleccionado. Hay que tener en cuenta la complejidad que supone resolver un problema de planificación, el cual es un problema de tipo PSPACE [19], lo cual implica que la elección del planificador puede ser muy importante, dependiendo del tipo de problemas que vayan a ser resueltos.

  A pesar de que la planificación automática trata de ser un sistema de res-

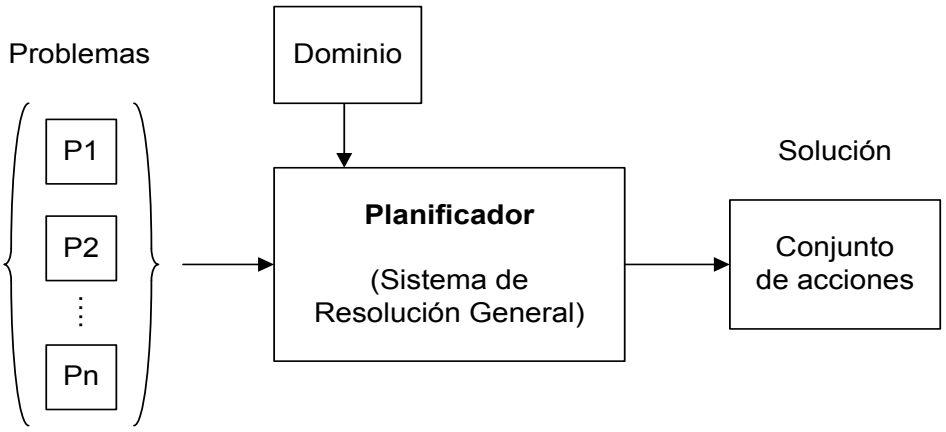


Figura 2.17: Ejemplo de funcionamiento de un planificador

olución general de problemas, en caso de que se intenten resolver problema del mundo real, es imposible modelar toda la información del entorno, no sólo debido a la cantidad de información sino a la complejidad que supone describir ciertos elementos que tal vez no influyan en el proceso de planificación. Por tanto es necesario realizar algunas asunciones con el fin de poder resolver problemas, utilizando estas técnicas:

- Conjunto de estados finitos: En entorno está formado por un conjunto finito de estados, entre los cuales el planificador el capaz de transitar mediante la aplicación de las acciones.
- Entorno completamente observable: El sistema posee un conocimiento completo de todos los elementos del entorno en un determinado estado.
- Entorno Determinista: La ejecución de una acción produce una transición a un único estado.
- Entorno estático: Las modificaciones en el entorno sólo se producen cuando se ejecuta una acción.
- Metas restringidas: El objetivo del planificador es obtener un conjunto secuencial de acciones denominado plan, que permitan transitar de un estado

denominado inicial a otro denominado final en el cual todas y cada una de las metas sean ciertas.

- Planes secuenciales: La solución generada por el planificador estará formada por un plan secuencial de acciones que permitirán transitar desde el estado inicial al estado meta.

- Tiempo implícito: Las transiciones entre los estados son instantáneas debido a que las acciones no tienen duración.

Con el fin de conseguir mucho mayor realismo en el proceso de planificación algunas de estas suposiciones puedes ser relajadas o eliminadas por completo. En el proyecto descrito en este documento a la hora de aplicar el algoritmo de planificación se asumen como ciertas todas estas suposiciones, aunque muchas de ellas no sean ciertas, por lo tanto la solución obtenida por el planificador no será un plan que pueda ser ejecutado de forma completa, ya que pueden darse situaciones en las cuales parte del plan debe ser calculado de nuevo debido a las variaciones que puedan producirse en el entorno durante el proceso de planificación y/o ejecución de las acciones.

### 2.3.2. Planificación Clásica

La planificación clásica puede considerarse como el proceso de búsqueda de un conjunto secuencial de acciones que deben ejecutarse desde un estado inicial, para llegar a un estado, que denominaremos final, en un entorno determinista y del que se posee conocimiento completo. En la figura 2.19, se puede observar el conjunto de acciones que ejecutaría un rover P3DX, para desplazarse desde su estado origen, hasta el estado marcado como meta. La información referente al mundo se encuentra representada mediante una cuadrícula de 6x4, que es totalmente conocida por el rover, es decir el conoce la localización de los obstáculos y de las posibles transiciones que pueden realizarse entre los distintos puntos que forman el mundo.

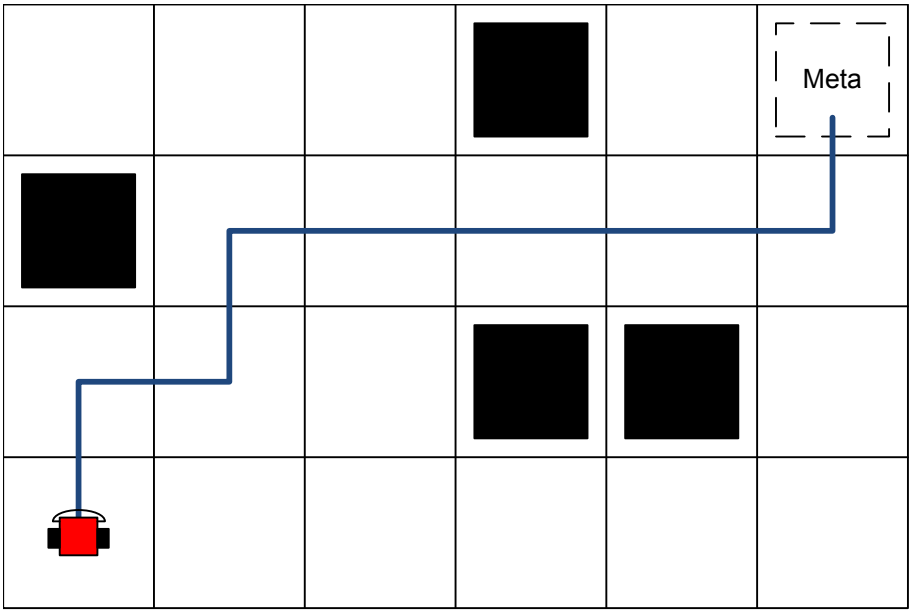


Figura 2.18: Ejemplo de un problema del mundo de los rovers.

#### 2.3.2.1. Modelo Conceptual

Como se describío anteriomente en al apartado anterior, para poder aplicar la planificación en necesario definir un modelo conceptual, el cual en la planificación clásica puede definirse como un sistema de transición entre estados, determinista y donde la información del sistema es siempre conocida, ya que las distintas transiciones entre los estados son las únicas que producen variaciones en el entorno. De forma más precisa podría definirse mediante el conjunto representado en la ecuación 2.4.

$$\Sigma = (S, A, \gamma) \tag{2.4}$$

- S es el conjunto finito de estados en los cuales puede encontrarse el sistema.
- A es el conjunto finito de acciones que pueden ser aplicadas sobre el sistema y que siempre producirán una transición a un único estado.
- $\gamma$ es la función de transición entre estados, de forma que $\gamma : SxA \rightarrow 2^S$.

#### 2.3.2.2. Lenguaje de representación

La representación de la información tal vez sea uno de los elementos más importante en cualquier proceso de razonamiento automático, debido a que le confiere de una mayor o menor expresividad teniendo en cuenta la granularidad de la representación utilizada. Desde sus orígenes, la planificación automática y en concreto la planificación clásica ha utilizado lenguajes basados en **lógica de predicados**, siendo una de las primeras representaciones utilizadas STRIPS (STandford Research Institute Problem Solver), desarrollados por los investigadores Fikes and Nilson a comienzos de los años 70 [8]. En este lenguaje un problema es representado mediante una tupla de la forma (P, A, I, M), donde:

- P es el conjunto finito de predicados que pueden ser ciertos en un estado, es decir, un representación global de todas los predicados que pueden ser utilizados para describir las características del entorno. Donde cada estado estará formando por un conjunto de predicados, de forma que lo que no aparece explícitamente representado es considerado como falso, teniendo en cuenta la textbfsuposición del mundo cerrado.

- A es el conjunto de operadores o acciones que pueden ser aplicados para producir una transición entre dos estados. Cada una estas acciones puede ser definida como una cuádrupla $(\alpha, \beta, \sigma, \gamma)$, donde $\alpha$ y $\beta$, se corresponde con las condiciones que deben ser verdaderas y falsas respectivamente para que pueda ser aplicada la acción, $\sigma$ representa los predicados que serán añadidos al estado, se harán ciertos, y $\gamma$ los que serán eliminados, es decir se harán falsas.

- I es el conjunto de predicados que son verdaderos en el estado inicial.

- M es la especificación del estado final o meta, el cual puede ser definido por una dupla (N,M), que indica los predicados que tienen que ser verdaderos y/o falsos para que un estado pueda ser considerado como meta.

Teniendo en cuenta esta representación, la solución a un problema podría representarse como un conjunto de acciones que pertenezcan al conjunto A, y que ejecutadas secuencialmente son capaces de transitar desde el estado I, hasta un estado en el cual se cumplan las metas definidas por M.

Varias décadas después de la aparición de STRIPS, apareció ADL (Action Description Language) [20], mediante el cual se ampliaron las capacidades de representación que STRIPS, debido a que mediante éste no era posible representar muchos elementos. Entre las más importantes extensiones que fueron incluidas, se encuentran:

- Utilización de cuantificadores.
- Utilización de la operación de igualdad.
- La aparición de efectos condicionales.
- Definición de tipos para las variables.

Pero fue a finales de la década de los 90, durante la competición de planificación de 1998, cuando apareció PDDL (Planning Domain Description Language) [21], el cual surgió como una necesidad para poder estandarizar la definición de dominios para los participantes. Actualmente, PDDL es el lenguaje estándar utilizado en planificación automática, el cual es de nuevo una extensión de su predecesor ADL, con un conjunto de nuevas extensiones, las cuales se van ampliando con cada nueva competición de planificación (IPC). En la figura 2.19 se presenta la descripción parcial del dominio del **mundo de los rovers** en PDDL.

Como se puede observar la descripción del dominio, se encuentra formada por algunos de los elementos anteriormente descritos en la definición de STRIPS. Se pueden observar la descripción de la estructura de todos los predicados que pueden ser utilizados, así como los distintas acciones u operadores que pueden ser aplicados. Como ya indicamos en la introducción de este apartado, la descripción de un problema PDDL se encuentra dividida en dos ficheros, uno que representa

```
(define (domain Rover)
(:requirements :typing)
(:types rover waypoint store camera mode lander objective)

(:predicates (at ?x - rover ?y - waypoint)
             (at_lander ?x - lander ?y - waypoint)
             (can_traverse ?r - rover ?x - waypoint ?y - waypoint)
             (store_of ?s - store ?r - rover)
             (calibration_target ?i - camera ?o - objective)
             (on_board ?i - camera ?r - rover)
)

(:action navigate
:parameters (?x - rover ?y - waypoint ?z - waypoint)
:precondition (and (can_traverse ?x ?y ?z) (available ?x) (at ?x ?y)
             (visible ?y ?z))
:effect (and (not (at ?x ?y)) (at ?x ?z))
)

(:action sample_soil
:parameters (?x - rover ?s - store ?p - waypoint)
:precondition (and (at ?x ?p) (at_soil_sample ?p)
(equipped_for_soil_analysis ?x) (store_of ?s ?x) (empty ?s))
:effect (and (not (empty ?s)) (full ?s) (have_soil_analysis ?x ?p) (not
        (at_soil_sample ?p)))
)
)
```

Figura 2.19: Ejemplo de un dominio en PDDL

el dominio que es el que se muestra en la figura 2.19 y otro que representa el problema a resolver, que se presenta en la figura 2.20.

Las acciones en PDDL varían en algunos elementos con respecto a STRIPS, en la figura 2.21 se presenta un ejemplo. En ellas se incluyen las variables que serán utilizadas para ejecutar la acción, indicándose sus tipos; las condiciones que deben darse para que pueda ejecutarse la acción; y los efectos que producirá la ejecución de la acción, que consistirán en la adicción o eliminación de predicados.

#### 2.3.2.3. Algoritmos

Es necesario aplicar una técnica de resolución para conseguir resolver un problema. En el caso de la planificación automática es posible aplicar dos tipos de técnicas. La primera de ella consiste en realizar una búsqueda en el espacio de estados mediante la cual encontrar un plan secuencial. La mayor parte de

```
(define (problem problem) (:domain Rover)
(:objects
        general - Lander
        rover0 - Rover
        rover0store - Store
        waypoint0 waypoint1 waypoint2 - Waypoint
        )
(:init
        (visible waypoint1 waypoint0)
        (visible waypoint0 waypoint1)
        (visible waypoint2 waypoint1)
        (visible waypoint1 waypoint2)
        (at_soil_sample waypoint2)
        (at_lander general waypoint0)
        (channel_free general)
        (at rover0 waypoint1)
        (available rover0)
        (store_of rover0store rover0)
        (empty rover0store)
        (equipped_for_soil_analysis rover0)
        (can_traverse rover0 waypoint0 waypoint1)
        (can_traverse rover0 waypoint1 waypoint0)
        (can_traverse rover0 waypoint1 waypoint2)
        (can_traverse rover0 waypoint2 waypoint1)
)

(:goal (and
        (communicated_soil_data waypoint2)
      )
)
)
```

Figura 2.20: Ejemplo de un problema del mundo de los rovers en PDDL.

los planificadores desarrollados hasta la fecha están basados en esta técnica. La segunda consiste en realizar una transformación sobre el problema para convertirlo en otro tipo de problema y utilizar una técnica de resolución mucho más precisa. Siendo el principal exponente de este tipo es la planificación SAT (Planning as boolean satisfiability), desarrollada por Kautz and Selman en 1992 [22].

Todos los algoritmos de búsqueda utilizados en los planificadores se basan en la realización de una búsqueda en un grafo finito. En la mayor parte de los casos, los nodos de este grafo se corresponden con estados del mundo, donde los arcos pueden considerarse como las acciones que producen la transición de uno a otro

```
(:action sample_soil
        :parameters (?r - rover ?w - waypoint)
        :precondition
                (and
                        (at_rover ?r ?w)
                        (at_soil_sample ?w)
                        (equipped_for_soil_analysis ?r)
                        (have_data_store ?r)
                )
        :effect
                (and
                        (have_soil_analysis ?r ?w)
                        (not (at_soil_sample ?w))
                )
)
```

Figura 2.21: Ejemplo de una acción del mundo de los rovers en PDDL.

estado. Existen también otras representaciones de los nodos del grafo incluyendo otra información, como por ejemplo la utilizada en el planificador GraphPlan, desarrollado por Blum and Furst en 1997 [23].

Con independientemente de la información almacenada en los distintos nodos que forman el grafo, todos los algoritmos de búsqueda utilizados en la mayor parte de los planificadores, pueden ser descritos mediante dos características básicas, mediante las cuales se define la manera que utilizan para recorrer el espacio de búsqueda.

- Dirección: Esta característica se corresponde con la dirección del proceso de búsqueda, es decir **hacia delante** si se realiza la búsqueda desde el estado inicial hasta un estado meta, o **hacia atrás** de forma que el algoritmo comenzaría buscando desde un estado en el que las metas fueran ciertas hasta el estado inicial.

- Selección: Consiste en la manera en la cual es seleccionado el siguiente nodo a explorar. En las etapas iniciales de la inteligencia artificial se utilizaban algoritmos de **fuerza bruta**, los cuales realizaban una exploración del espacio

de búsqueda en profundidad o anchura. Esto suponía que en muchos casos no se encontrara solución o el tiempo para conseguirlo fuera extremadamente elevado. Pero en los siguientes años surgieron algoritmos basados en funciones heurísticas que utilizaban conocimiento extra del problema a resolver, para seleccionar el mejor candidato a explorar, como por ejemplo el algoritmo A* [24], la aparición de este tipo de algoritmos influyó de manera significativa al proceso de desarrollo de planificadores, dando lugar a la aparición de los primeros planificadores heurísticos, como el planificador FF, desarrollado por Hoffmann en 2001 [25].

#### 2.3.2.4. Sayphi

Para el desarrollo de este proyecto se ha seleccionado el planificador Sayphi [26], el cual ha sido desarrollado por el grupo PLG (Planning and Learning Group) de la Universidad Carlos III de Madrid. Sayphi es un planificador automático que integra diferentes técnicas de aprendizaje automático que pueden ser aplicadas a la planificación. Además incluye un planificador de tipo Fast Foward, similar al que utiliza el planificador FF [25] y que permite la utilización de múltiples algoritmos de búsqueda, como por ejemplo, el algoritmo A* [24].

### 2.3.3. Conclusiones

La planificación automática es una técnica de la inteligencia artificial que nos permite la resolución de problemas mediante la generación de un plan secuencial de acciones. Este proceso de resolución es una importante característica la cual se adaptaba muy bien a las necesidades del proyecto descripto en este documento, ya que nos permite describir el conjunto de acciones necesario para resolver un problema y aplicarlas sobre un conjunto de robots de forma que puedan resolver tareas complejas mediante la aplicación de tareas muy sencillas, como por ejemplo, moverse o girar.

# Capítulo 3

# Arquitectura del controlador

## 3.1. Introducción

En este apartado del documento se realiza una descripción detallada de la arquitectura desarrollada para este proyecto fin de carrera, denominada $AMaC^2$ (Arquitectura multi Agente mediante control centralizado) . El controlador presentado en este documento puede definirse mediante tres características básicas, las cuales describen su funcionamiento y fundamentan su estructura.

- **Híbrida**, la principal característica del controlador deriva de los distintos procesos que serán utilizados para generar los distintos comportamiento inteligentes que los robots serán capaces de generar. Teniendo en cuenta los distintos tipos de arquitectura que han sido descritas en el capítulo anterior, lo más conveniente será utilizar una arquitectura de tipo híbrido, que nos permite utilizar la planificación automática como técnica deliberativa para producir los distintos comportamiento de alto nivel o racionales, que junto con un grupo de sistema reactivos definirá las acciones de bajo nivel (reflejos, comportamiento primarios, etc) que podrán realizar los robots y que ejecutadas en un determinado orden de forma coordinada producirán que los robots sean capaces de producir comportamientos que puedan ser considerados inteligentes.

- **Multi-agente**, teniendo en cuenta que nuestro objetivo se centra en la utilización de múltiples robots que sean capaces de producir comportamientos colaborativos con el fin de resolver problemas complejos de forma coordinada, es necesario dotar de un grado mayor de distribución a la arquitectura. Esto nos permitirá utilizar sistemas de organización centralizada, que minimizará los costes computaciones en cada robot y mejorará el proceso de sincronización entre ellos, aunque suponga algunas desventajas teniendo en cuenta los elevados costes de comunicación que puede suponer según aumenta el número de robots que forma parte del sistema.

- **Modular**, el controlador tiene la capacidad de controlar múltiples robots de forma sencilla, esto implica que será necesario definir un sistema modular para añadir o eliminar nuevos robots de forma sencilla, más teniendo en cuenta que los robots que utilizaremos serán distintos, lo que implicará que utilizarán distintos controladores. Para permitir al controlar esto, se ha seleccionado el framework de desarrollo Microsoft Robotic Developer Studio [27] [28], el cual nos permite construir unidades de procesamiento independientes denominadas DDS (Decentralized Software Services), las cuales pueden ser considerados como módulos independientes que se comunican mediante SOAP.

En la figura 3.1, se puede observar el modelo de razonamiento de $AMaC^2$, esta puede ser considerada como una arquitectura híbrida y multi-agente, dispuesta en tres capas, las cuales interactúan entre sí de forma vertical. Como se puede observar esta arquitectura de control tiene muchas similitudes con la Arquitectura Atlantis [15] descrita en el capitulo 2 de este documento, con la salvedad de que aquella arquitectura está definida para ser utilizada por un único robot. En cambio en este caso la arquitectura ha sido definida para que sea utilizada por más de un robot sin que sea necesario realizar un número elevado de modificaciones. A nivel general las funcionalidades ofrecidas por cada una de las capas son las siguientes:

- Capa deliberativa: Esta es la capa de más alto nivel y se encarga de construir

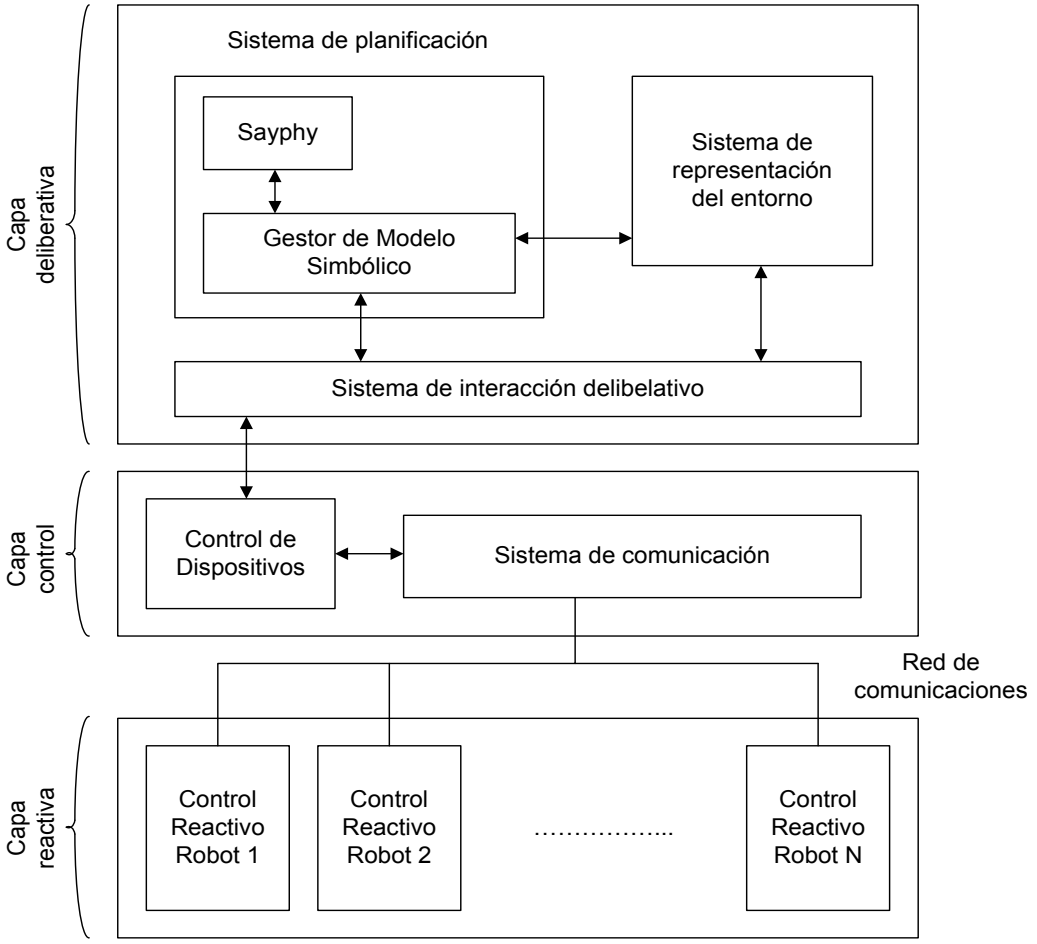


Figura 3.1: Modelo de razonamiento de la arquitectura.

el conjunto de acciones que deben seguir los robots para poder resolver el problema, está formada por un planificador y un conjunto de sistema que representan la información del entorno, a partir de una definición inicial y de la información obtenida a través de los sensores.

- Capa control: Esta capa de comunicación, se encuentra parcialmente formado por los sistema de comunicación ofrecidos por el framework de desarrollo de MRDS, el cual ha sido utilizado para producir la comunicación entre el sistema de control y los distintos robots.

- Capa reactiva: Esta capa, la de más bajo nivel, es la que está en contacto con los sensores y actuadores de los distintos robots que forman parte de

la organización. En ella se encuentran los controladores reactivos de cada uno de los robots, los cuales interactúan con la capa de comunicaciones, a través de que reciben ordenes desde el sistema de control central y envían información referente a las acciones realizadas e información relevante para el sistema que ha sido obtenida a través de los sensores.

## 3.2. Roles de ejecución

Teniendo en cuenta la estructura de la arquitectura $AMaC^2$, esta puede ser dividida en fragmentos que pueden ser repartidos entre distintos dispositivos, de forma que los distintos dispositivos podrán sólo ejecutar alguna de las capas de arquitectura, fomentando su naturaleza distribuida y permitiendo teóricamente explotar su capacidad de escalabilidad. Por lo tanto, teniendo en cuenta las distintas capas y las relaciones que existen entre ellas, se pueden definir tres posibles roles que podrán adoptar los distintos dispositivos que sean utilizados para desplegar la arquitectura.

- Coordinador: Un dispositivo adopta este rol cuando ejecuta las capas deliberativo y de control de la arquitectura. Bajo este rol el dispositivo gestiona los procesos de razonamiento, sincronización y comunicación entre los distintos dispositivos que se ejecutan bajo el rol de actuador. Este rol es **único** y actualmente sólo puede ser ejecutado por un sólo dispositivo, el cual no tiene por que ser un robot, pudiendo ser una centralita de control o un satélite.

- Actuador: Un dispositivo adopta este rol cual ejecuta una de las capas reactivas de la arquitectura. Bajo este rol los dispositivos gestionan las funciones básicas de los robots que son controlados por la arquitectura.

- Global: Este rol sólo aparece cuando un único dispositivo se encarga de ejecutar todas las capas de la arquitectura, independientemente del número de robots que estén siendo controlados.

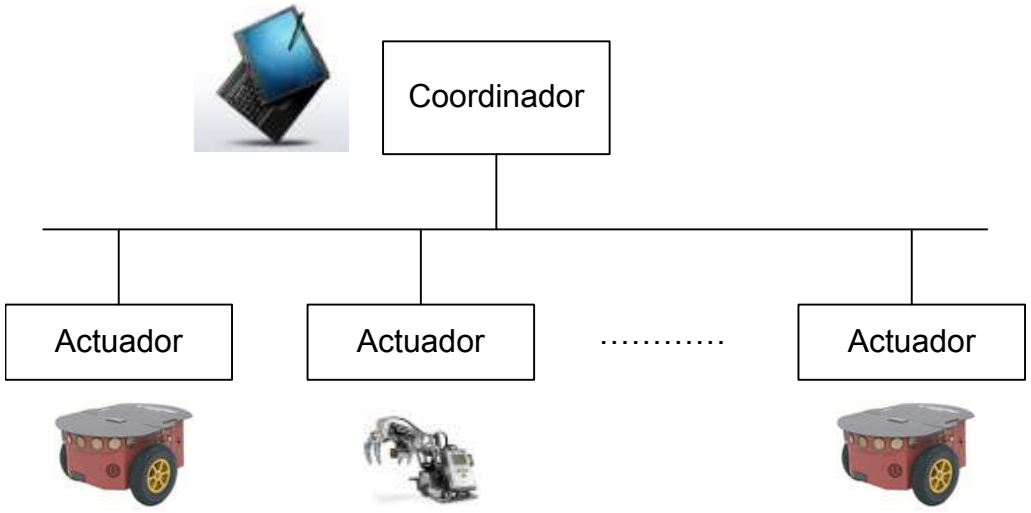


Figura 3.2: Distribución de roles.

Teniendo en cuenta la estructura de la arquitectura $AMaC^2$, podrían darse múltiples configuraciones. En la figura 3.2 se presenta un ejemplo de una de las posibles configuraciones donde cada uno de los robots tendría una CPU propia, que ejecutaría una capa reactiva bajo el rol **actuador** y existiría una centralita independiente que se ejecutaría las capas deliberativa y de control bajo el rol de **coordinador**.

## 3.3. Arquitectura detallada

A la hora de implementar sobre un determinado *framework* la arquitectura de control $AMaC^2$, es necesario realizar ciertas modificaciones. En este caso al haber sido seleccionado el *framework* de desarrollo *Microsoft Robotic Developer Studio*, es necesario diseñar una estructura constituida por los módulos que el ofrece, que en este caso son los DSS (Decentralized Software Services), los cuales pueden ser considerados como módulos intercambiables que se comunican mediante servicios web. La ventaja de utilizar estas unidades es su sencillez de programar una vez conocido su funcionamiento y las facilidades que existen para poder intercambiar los módulos, lo cual es uno de los elementos que se buscaba en el

sistema de control. A continuación, en la figura 3.2 se presenta un diagrama de cómo queda integrado el sistema de control mediante los elementos que ofrece el *framework*.

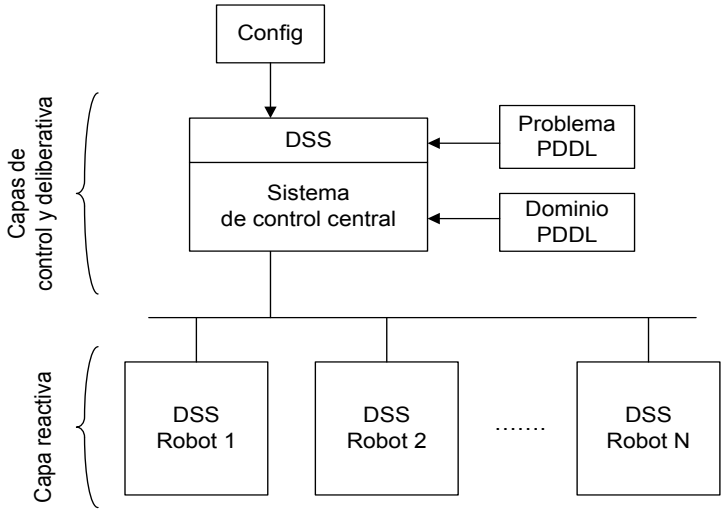


Figura 3.3: Modelo de razonamiento implementado mediante MRDS.

Como se puede observar en la figura 3.3, el sistema estaría formado por un DSS (Decentralized Software Services), en el cual se implementarían las capas deliberativa y de control, y por un conjunto de DSS, uno por cada robot, en los cuales se implementarían los elementos correspondientes del nivel reactivo, cabe destacar que en cada uno de los servicios deben implementarse los sistemas de comunicación que son necesarios para producir la interacción entre los distintos elementos y el sistema de control centralizado. Para obtener más información referente a MRDS, consultar el Apéndice A de este documento.

### 3.3.1. Sistema de Control Central

En este apartado se realiza una descripción de los distintos elementos que forman el servicio de control centralizado, en el cual se engloban el sistema de planificación, la gestión de mapas y el sistema de control. Este sistema se corre-

sponde con las capas deliberativas y de control del sistema, lo que implica que el sistema de razonamiento de alto nivel se encuentra desarrollado en él.

Como se puede observar en la figura 3.4, el sistema de control central que se corresponde con la capa deliberativa, está formado por cuatro subsistemas, los cuales interactúan entre si para generar los planes de tareas que resuelvan el problema. A continuación se describen las funcionalidades básicas de cada uno de ellos.

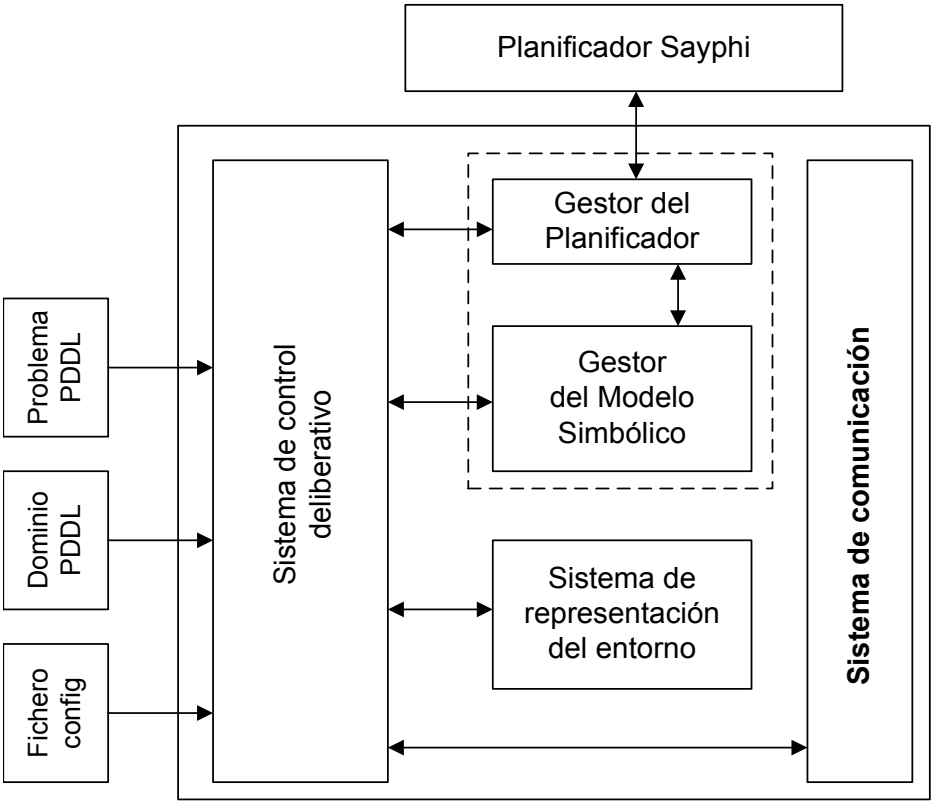


Figura 3.4: Estructura del sistema de Control Central

#### 3.3.1.1. Sistema de planificación

El primer subsistema es el de planificación, y puede ser considerado como el motor de razonamiento principal del sistema, ya que se encarga de la generación de los planes de acciones y la gestión del sistema de replanificación. Este se encuentra

formado por tres módulos:

- Gestor del Planificador: En este módulo se gestionan los elementos relacionados con el planificador, es decir se realiza la preparación de los ficheros necesarios para la generación del plan y se gestiona la salida generada por el planificador, comprobándose si se han producido errores o en caso contrario almacenándose las acciones a realizar para que puedan ser enviadas las ordenes a los distintos robots y resolver el problema, en el mundo real.

- Gestor del modelo simbólico: En este módulo se almacena toda la información referente al dominio y al entorno utilizando el lenguaje simbólico utilizado por el planificador, además de las distintas acciones que deben ser aplicadas y el estado en el cual se encontraba el entorno previamente a la aplicación de cada acción. Este módulo no ofrece la posibilidad de gestionar de forma sencilla la replanificación y en un futuro sería posible aplicar procesos de análisis de la acciones con el fin de permitir el paralelismo a nivel de los distintos robots. Además también nos ofrece un sistema de parcial de monitorización ya que permite comprobar si el estado generado en el mundo tras la aplicación de una acción, se corresponde con su homologo en representación simbólica.

- Planificador: Este elemento se corresponde con el planificador seleccionado para la resolución de los problemas, que en este caso ha sido el planificador Sayphi. Ha sido definido fuera de la arquitectura debido a que podría ser seleccionado otro planificador para realizar la generación de los planes de actuación.

#### 3.3.1.2. Sistema de representación del entorno

Este subsistema se corresponde con una representación intermedia o abstracta de la información contenido en el gestor del modelo simbólico y es la utilizada para realizar las actualizaciones procedentes de las variaciones que se producen en el entorno según se van ejecutando las acciones. Esta segunda representación

es más sencilla de manipular que la representación simbólica y además nos ofrece de forma sencilla una capa de monitorización de las acciones realizadas por los distintos dispositivos y las variaciones que se producen en el entorno. Aunque es más sencilla de manipular tiene algunas complicaciones, destacando que es **dependiente del dominio**, lo cual produce que deba ser manipulada en caso de añadir nuevos predicados u acciones al dominio que deban ser reconocidas y por tanto transformadas a esta representación. Como punto fuerte es la simplicidad de su estructura la cual se basa en dos elementos:

- Matriz de entorno: Esta matriz define todos los posibles puntos o waypoints definidos en el entorno, representando las transiciones que pueden realizarse entre ellos y la visibilidad que existe de unos a otros.
- Lista de objetos: Esta lista almacena todos los elementos que se encuentran en el entorno, indicando su posición, con respecto al sistema de coordenadas que ha sido definido en la Matriz de entorno, así como las relaciones que existen entre los distintos objetos, es decir uno está contenido en otro, etc.

Como se describió anteriormente, a pesar de la facilidad que ofrece a la hora de manipular la información, este sistema de representación es dependiente del dominio. Esto supone que en el caso de que sea necesario realizar modificaciones en el dominio, por ejemplo para incluir un mayor realismo, es necesario modificar el código fuente del sistema de representación.

#### 3.3.1.3. Sistema de control deliberativo

El subsistema de control deliberativo se ocupa de la gestión de toda la capa de razonamiento, sus principales funciones son las siguientes:

- Gestión de los fichero de entrada, en los cuales se especifica la configuración del sistema de control, la información conocida del entorno y que será utilizada inicialmente por el planificador especificado en PDDL y el problema resolver especificado también en PDDL.

- Control del sistema de planificación y del sistema de representación del entorno, controlando el proceso de comunicación que se produce entre ellos.

- Control del flujo de información entre los distintos elementos que forman la arquitectura del control, la cual debe ser coordinada y sincronizada de manera ordenada.

Debido a la estructura del *framework*, el cual está basado en eventos, es necesario la utilización de una estructura de control que nos permita sincronizar en qué momento se tiene en cuenta la información de un evento, por ejemplo la llegada de la información del sónar o cuando debe ejecutarse la lectura de un fichero, etc. Por lo tanto, para poder controlar de forma ordenada este tipo de procesos se ha desarrollado un autómata finito determinista mediante el cual se gestiona una parte del flujo de ejecución del sistema de control deliberativo, este se describe en la figura 3.5, en la cual se presenta un autómata formado por 6 estados:

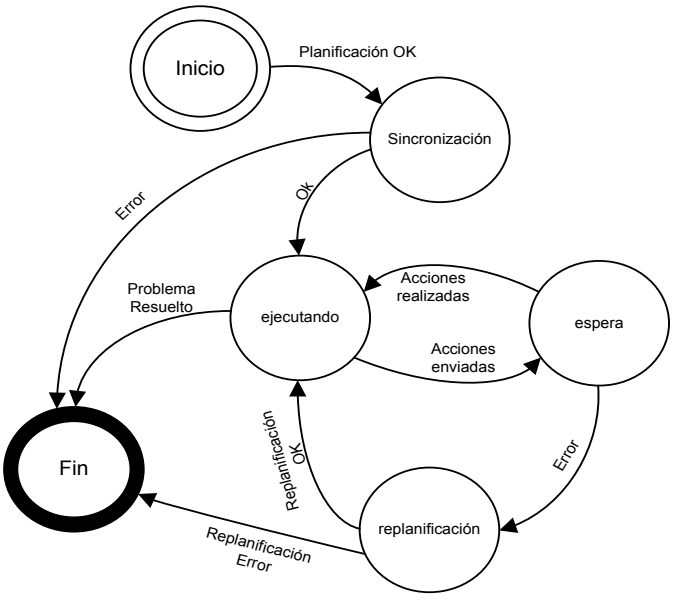


Figura 3.5: Autómata finito de control

- Inicio: Es el estado inicial del sistema, en el se realizan las distintas conexiones entre los DSS, se cargan los ficheros de configuración y se construyen los modelos simbólico y abstracto de la información contenida en los ficheros PDDL. Además se realiza la ejecución del planificador para el problema introducido como parámetro y en caso de que se encuentre la solución se pasa al estado siguiente.

- Sincronización: En este estado se produce la asignación de los roles de los distintos robots, es decir, se asigna los identificadores utilizados por el planificador a cada uno de los agentes físicos.

- Ejecución: Es este estado comienza el proceso de ejecución de tarea, donde el sistema de control deliberativo envía las acciones a realizar a los distintos robots.

- Espera: Este estado es un estado de sincronización, donde el sistema de control espera a que los robots realicen las acciones e información con el resultado de la acción. En caso de que sea correcta se vuelve a estado de ejecución, y en caso contrario se vuelve al estado de Fin.

- Replanificación: Es el estado en el cual se produce el proceso de replanificación, en caso de encontrar un nuevo plan. El sistema vuelve al estado de ejecución y prosigue con la resolución del problema; en caso contrario se transita al estado Fin, debido a que no es posible resolver el problema.

- Fin: Es el estado final, al cual se llega cuando ha sido resuelto el problema en el entorno físico, o se ha producido un error al intentar resolver el problema.

### 3.3.2. Sistema de control reactivo

Los sistemas de control desarrollados para los diferentes robots que serán utilizados en este proyecto, están formado por dos elementos:

- Sistema de interacción: Este se encarga de interactuar con los distintos sensores y actuadores que forman la estructura del robot, ofreciendo un interfaz interactivo que se encarga de obtener y procesar la información obtenida a través de los sensores y enviar la información necesaria a los actuadores.

- Autómata de control: Este se encarga de regir el flujo de ejecución del robot, teniendo en cuenta la información obtenida a través de los distintos sensores y el sistema de control central de la arquitectura.

#### 3.3.2.1. Sistemas de interacción

El sistema de interacción desarrollado para los diferentes robots utilizados en este proyecto, está formado por un conjunto de operaciones que interactúan entre sí, teniendo en cuenta una serie de eventos que se van produciendo durante la ejecución del sistema de control y una serie de acciones que se producen como respuesta a dichos eventos. A continuación se presenta los sistema de interacción diseñados para cada uno de los robots que han sido seleccionados para el desarrollo de este proyecto. Teniendo en cuenta el modo de funcionamiento de los distintos sistemas de interacción que han sido desarrollados, se pueden distinguir dos tipos de procesos:

- Sensoriales: Son aquellos procesos (Golpeo Bumper, Golpeo Múltiples Bumper y Detección Objeto) de naturaleza totalmente reactiva, ya que interactúan en todo momento con la información recogida a través de los sensores, procesándola tras la recepción, aunque esta sea posteriormente ignorada por el sistema de control.

- Motrices: Son aquellos procesos (Mover, Detener y Girar) que son producidos por la recepción de un mensaje o estímulo racional, enviado por el sistema de control central. Estos procesos son parcialmente reactivos, ya que su ejecución se puede ver alterada o anulada por la recepción de información de los procesos sensoriales. Por ejemplo, si el sónar del robot P3-DX detecta

un objeto en una posición que impide al robot moverse, éste debe volver a su posición inicial y comunicárselo mediante un mensaje al controlador central.

En la figura 3.6 se presenta un diagrama que describe el proceso de funcionamiento del sistema de interacción del robot P3-DX.

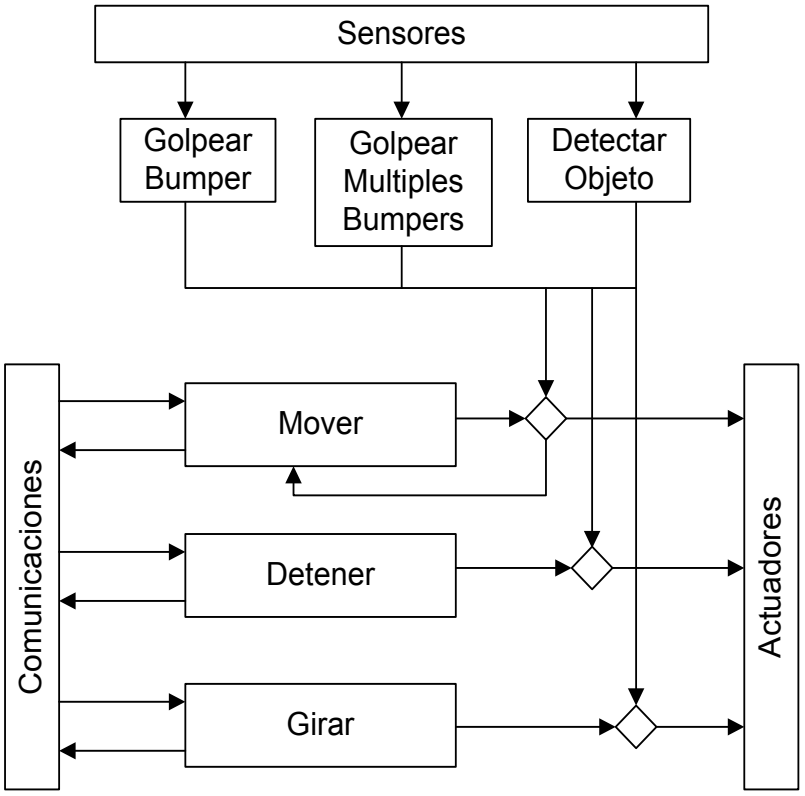


Figura 3.6: Estructura interactiva del sistema de control del rover

Como se puede observar en la figura 3.6, el sistema de control esta formado por un conjunto de procesos que se ejecutan de forma concurrente, mediante la activación de estos por parte de los sensores o los mensajes de control que han sido enviados por controlador central. El funcionamiento del sistema es muy sencillo, se basa en la ejecución de un conjunto de acciones (Mover, Detener y Girar) previa recepción de un mensaje enviado por el sistema de control central, durante el proceso de ejecución de las acciones pueden surgir complicaciones que impidan la realización total de la acción o la detención temporal de la ejecución, lo cual es detectado por los sensores y debe ser resuelto en la medida de los posible por

el controlador local del robot, en caso de no poder ejecutar de forma completa la orden recibida mediante el sistema de comunicación, el sistema debe abortar la ejecución, volver a la posición previa a la ejecución de la acción y enviar la información detectada al sistema de control central de forma que genere una alternativa.

En la figura 3.7 se presenta el diagrama que describe el proceso de funcionamiento del sistema de interacción del robot Lego NXT.

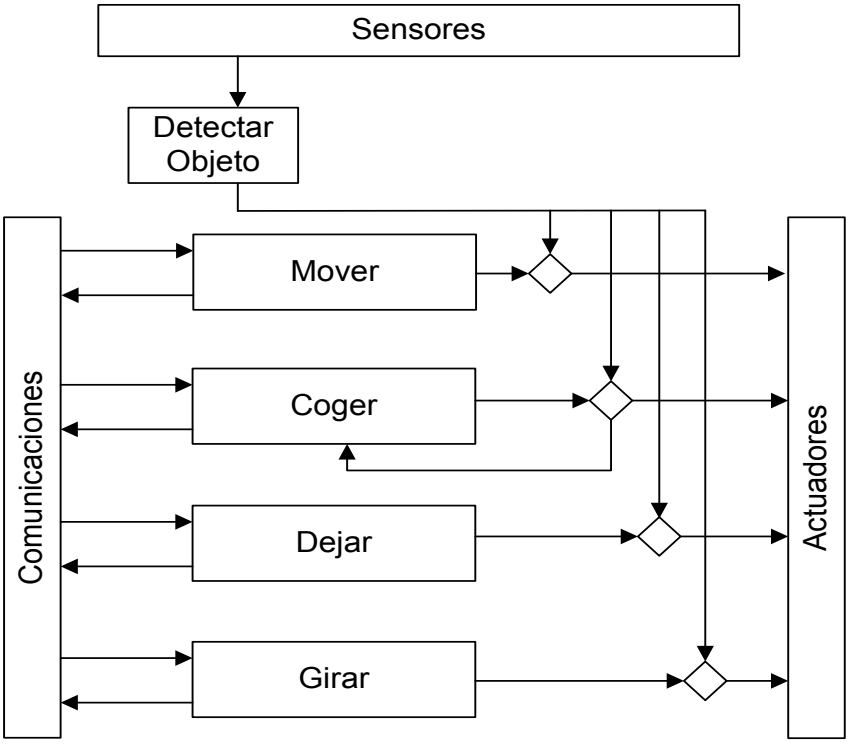


Figura 3.7: Estructura interactiva del sistema de control del brazo

Como se puede observar en la figura 3.7 el sistema de control esta formado por un conjunto de procesos que se ejecutan de forma concurrente, mediante la activación de estos por parte de los sensores o los mensajes de control que ha sido enviados por el controlador central. El funcionamiento del sistema es muy sencillo, se basa en la ejecución de un conjunto de acciones (Girar, Detener, Coger y Soltar) previa recepción de un mensaje enviado por el sistema de control central. En este caso el control reactivo es mucho más sencillo que en el caso del P3-DX ya que

sólo se ha incluido un sensor de presión para detectar la recogida de los objetos, en caso de que el objeto no sea detectado cuando se produce la recogida, indica al robot que el objeto no se encontrada en la zona de recogida o que este se ha caído al intentar ser recogido por las pinzas, en ese caso el sistema debe informar al sistema de control central, finalizando de forma fallida el proceso de resolución del problema, debido a que no disponemos de ningún sistema que nos permita buscar los objetos, por ejemplo mediante un cámara de visión y en el caso de este fuera incluido sería dependiente del radio de actuación del brazo robótico, el cual en este caso es muy reducido.

#### 3.3.2.2. Autómata de control

Al igual que ocurría con el sistema de control central, el sistema de control reactivo de cada uno de los robots debe encontrarse también gobernado por un autómata finito que permita al robot abstraerse de información no útil y ofrezca un sistema sencillo de sincronización con el sistema de control central, en la figura 3.8, se presenta el autómata de control desarrollado para los sistemas de control reactivo de los robots.

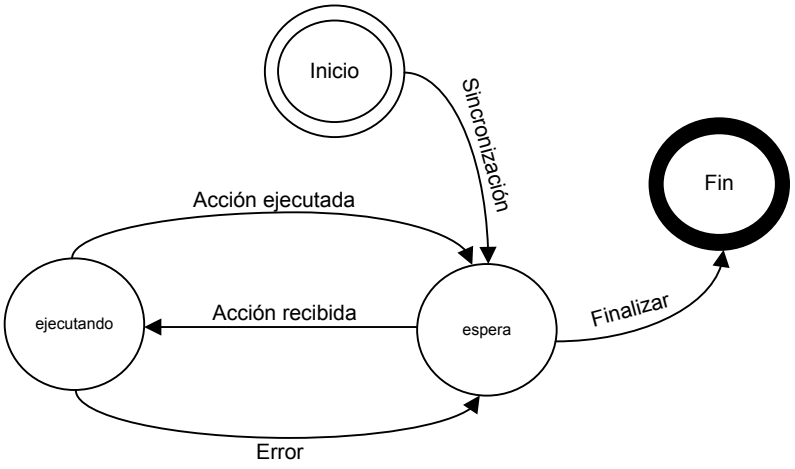


Figura 3.8: Autómata finito de control reactivo de robots

- Inicio: Es el estado inicial del robot, en el se realizan las distintas conexiones

entre los DSS que gestionan los sensores y actuadores del robot y el sistema de control central. Además este es un estado de espera en el cual el robot no utiliza la información obtenida a través de los sensores y espera a recibir el mensaje de sincronización para conectarse al sistema de control centralizado y recibir ordenes.

- Ejecución: En este estado se realiza el proceso de ejecución de las acciones enviadas por el sistema de control central. Es el servicio más complejo del robot, ya que en él deja de ser ignorada la información que es obtenida a través de los sensores, ya que la única información que es importante es aquella que se obtiene cuando el robot esta ejecutando una acción previamente indicada por el sistema de control central. En caso de que se obtenga información del entorno que no permita finalizar la acción solicita, el robot volverá a su estado anterior e indicará al sistema de control que se ha producido un error y este lo corrija si es posible.

- Espera: Este estado es un estado de sincronización, donde el sistema de control del robot espera a recibir nuevas acciones a realizar por parte del sistema de control.

- Fin: Es el estado final, en el cual se produce la desactivación de los sistemas de comunicación con el resto de servicios. La transición a este estado se produce por la recepción de un mensaje de fin, el cual se puede producir por que se haya logrado resolver el problema o por que se ha producido un error que evita la finalización del problema.

## 3.4. Comunicaciones

El proceso de comunicación entre los distintos elementos que forman el sistema de control se realiza mediante los servicios basados en SOAP, ofrecidos por el FrameWork MRDS 2008 R2. Estos permiten conectar los distintos sistemas, de forma sencilla utilizando paquetes de información. A pesar de su aparente ventaja

este método de comunicación no es tan eficiente como aparentemente parece, ya que sólo permite el envío de tipos básicos de datos, como enteros, flotantes y cadenas de caracteres, lo que supone que toda la información que debe ser enviada debe ser transformada en estos tipos de datos.

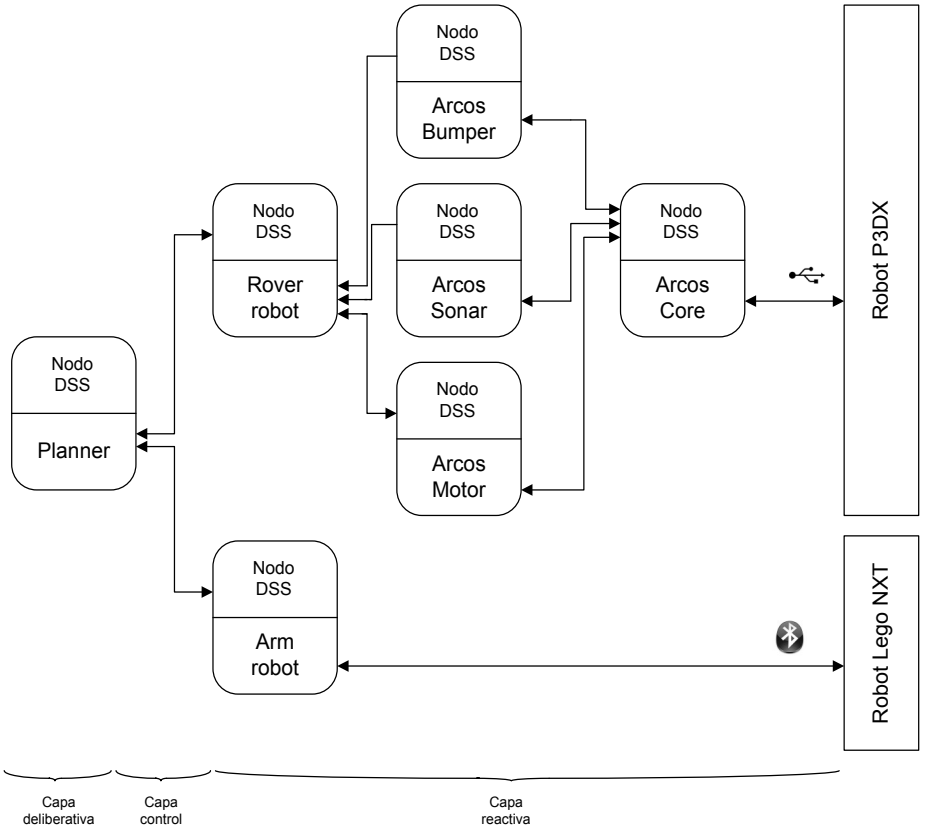


Figura 3.9: Comunicación entre servicios y dispositivos

En la figura 3.9, se presenta un diagrama de los distintos enlaces de comunicación que se producirán entre los distintos servicios que serán utilizados por el sistema de control y los distintos dispositivos. Atendiendo a este diagrama podemos definir dos tipos de comunicaciones.

- Comunicaciones internas: Estas comunicaciones se corresponden con todas las asociaciones que se realizan entre los distintos servicios que han sido desarrollados o incluidos. Son denominadas internas debido a que todas se

encuentra teóricamente distribuidas como nodos de un red de comunicaciones.

- Comunicaciones externas: Son la conexiones que se producen entre los servicios y los robots. En este caso existen dos comunicaciones de este tipo; la primera se realiza mediante un cable usb que se conecta entre uno de los puertos COM del robot P3-DX y un usb del ordenador donde se ejecutan los servicios de control del robot; y la segunda se realiza mediante *bluetooth* entre la centralita de control del robot Lego NXT y el ordenador donde se ejecuta el servicio de control del brazo robot.

Además, como se puede observar en el diagrama, la disposición de los servicios se corresponde con la estructura de la arquitectura, ya que los servicios han quedado distribuidos de forma jerárquica, la cual se asemeja en gran medida con la diferenciación por capas de la arquitectura presentada en la figura 3.1. Por último es necesario destacar que esta interconexión entre los distintos servicios descentralizados, permite la ejecución en máquinas o procesadores distintos, fomentando el paralelismo en la ejecución de lo procesos e interactuando entre ellos mediante la utilización de las redes de comunicación disponibles.

Teniendo en cuenta esta distribución, los ordenadores de abordo de los distintos robots ejecutarían de forma independiente los distintos servicios que forman el controlador y se comunicarían mediante la utilización de una red inalámbrica. En este caso no se a podido aplicar este tipo de distribución ya que sólo disponíamos de un único ordenador en el cual se ejecutaban de forma concurrente todos los servicios que formaban el sistema de control. Este sistema se asemeja en gran medida a la estructura organizativa y funcional que se puede observar en el cerebro, donde existen zonas especializadas en un conjunto de funciones y que se comunican con otras áreas mediante una red.

## 3.5. Planificación

Puesto que estamos utilizando una arquitectura híbrida, nuestro sistema de control además de contar con funciones reactivas, debe poseer un sistema de razonamiento avanzado, basado en la utilización de la inteligencia artificial. Como ya se indicó al comienzo de este capítulo, se decidío utilizar **planificación automática**, la cual se adapta muy bien a las necesidades del sistema desarrollado, ya que es capaz de construir un plan de acciones, en el cual existan procesos colaborativos entre distintos robots. A continuación se realiza una descripción del dominio que ha sido desarrollado para este proyecto, así como el funcionamiento del proceso de replanificación el cual ha sido utilizado en situaciones en las cuales no era posible llevar a cabo el plan generado inicialmente por el sistema de planificación.

### 3.5.1. Definición del dominio

En dominio utilizado para la resolución de problemas para este proyecto, ha sido el dominio de los **rovers**, este dominio fue diseñado para la generación de planes de acciones, mediante las cuales un rover pudiera realizar una serie de operaciones en un entorno completamente conocido. Su función principal consiste en la recogida de muestras y la realización de fotografías. Debido a que en este proyecto se decidió incluir un segundo robot, fue necesario realizar una serie de modificaciones llegando al dominio utilizado el proyecto el puede resumirse de la siguiente manera.

> El dominio consiste en la existencia de un conjunto de robots de dos tipos, excavadores y transportadores, los primeros son capaces de recoger rocas u objetos y a continuación cargarlos en la zona de carga de los robots de tipo transportador, los cuales son capaces de transportar objetos de una zona a otra del entorno y recoger pequeñas muestras de rocas y arena que se encuentren en el suelo, transmitiéndolas posteriormente a un punto de control.

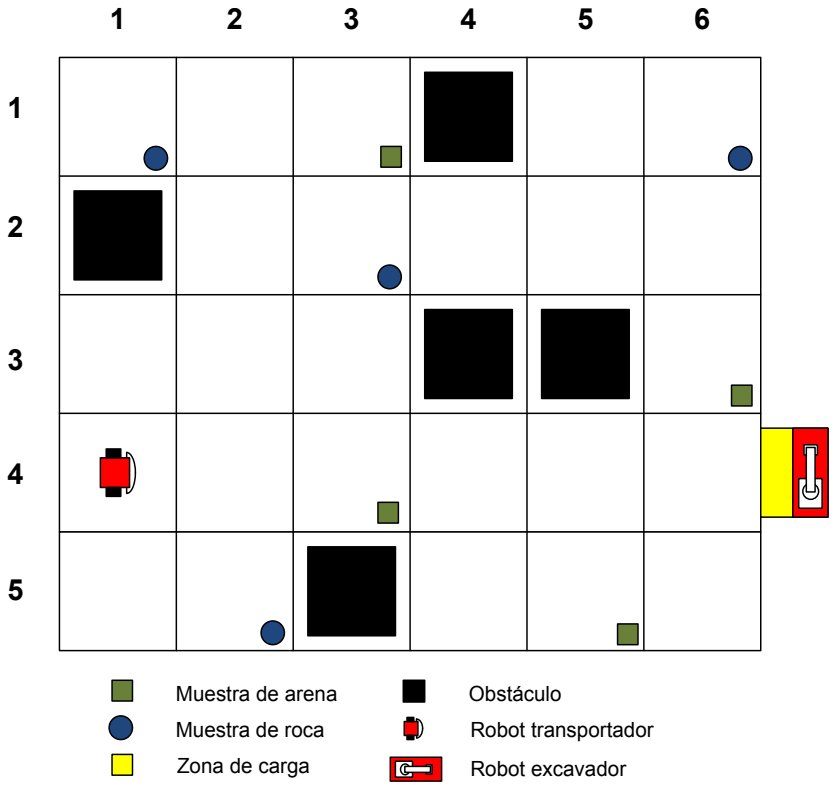


Figura 3.10: Descripción gráfica de los elementos principales del Dominio.

En la figura 3.10 se muestra un ejemplo visual del dominio, en el cual se pueden observar los distintos robots que forman parte del entorno, así como los distintos objetos que pueden encontrarse, como muestras de roca, arena, o objetos que impiden el paso del robot transportador. A continuación se realiza una descripción más completa de los distintos objetos.

- Robot transportador: Es un robot de tipo rover, el cual puede moverse entre los distintos *waypoints*. Además es capaz de transportar objetos y recoger muestra de arena y roca.

- Robot excavador: Es un robot de tipo excavador, es capaz de recoger objetos de la zona de excavación en la que se encuentra situado.

- Objeto: Es un objeto que ha sido recogido por un robot excavador y/o

transportado por un robot transportador.

- Muestra de roca: Es una muestra de roca que se encuentra en un *waypoint* y puede ser recogida por un robot transportador.

- Muestra de arena: Es una muestra de arena que se encuentra en un *waypoint* y puede ser recogida por un robot transportador.

En este dominio es posible aplicar distintas acciones para producir las transiciones entre los estados, las cuales son descritas a continuación:

- Navegar (navigate): Permite a un robot de tipo transportador moverse entre dos puntos del mapa, denominados waypoints. En el mundo real, los rovers se moverán entre los puntos centrales de los waypoint, ya que estos son considerados como cuadrados.

- Girar (turn): Permite a un robot de tipo transportador variar su orientación, entre cuatro posibles, representadas por los cuatro puntos cardinales (Norte, Sur, Este y Oeste). Se ha realizado esta representación teniendo en cuenta posibles ampliaciones que nos permitieran utilizar un compás, conociendo de forma precisa la orientación del robot.

- Coger objeto (pick_up): Permite a un robot de tipo excavador recoger un objeto. El objeto debe encontrarse en la zona de recogida.

- Dejar objeto (drop): Permite a un robot de tipo excavador soltar un objeto previamente recogido en la zona de carga de un robot de tipo transportador.

- Mover a posición de carga (move_loading): Permite a un robot de tipo excavador moverse a la zona de carga desde la zona de descarga. Para simplificar el proceso de gestión de objetos han sido definidas dos posiciones para el brazo; una que denominaremos de carga, en la cual el brazo puede recoger un objeto; y otra que llamaremos de descarga, en la cual el brazo tiene recogido un objeto y puede soltarlo.

- Mover a posición de descarga (move_uploading): Permite a un robot moverse a la zona de descarga.
- Tomar muestra de área (sample_soil): Permite a los robot de tipo transportador recoger muestras de arena que se encuentra en el mismo waypoint en el que se encuentra.
- Tomar muestra de roca (sample_rock): Permite a los robot de tipo transportador recoger muestras de roca que se encuentra en el mismo waypoint en el que se encuentra.
- Entrar en zona de carga (enter_charge_area): Permite a los robot de tipo transportador entrar en las zonas de carga donde se encuentran los robots excavadores.
- Salir de zona de carga (exit_charge_area): Permite a los robot de tipo transportador salir de las zonas de carga a las que han entrado previamente.
- Transmitir muestra de suelo (comunicate_soil_data): Permite transmitir la información obtenida tras el análisis de la muestra recogida de arena del suelo.
- Transmitir muestra de roca (comunicate_rock_data): Permite transmitir la información obtenida tras el análisis de la muestra recogida de roca del suelo.

Al final de este documento en el Anexo D, se incluye el código en PDDL, que ha sido desarrollado para la definición de este dominio.

### 3.5.2. Replanificación

Uno de los grandes problemas que se observaron en el sistema de control del robot P3-DX, es que si este tenía en cuenta los objetos que se movían a su alrededor cuando se encontraba en el estado de espera, podían producir que modificara su posición. Esto podía dar lugar a situaciones en las cuales el robot podría perderse. Inicialmente se planteo resolver este problema mediante un cálculo matemático

teniendo en cuenta las acciones del robot, pero luego se decidío simplificar más el sistema de control, teniendo en cuenta sólo aquellos elementos que le impedían realizar la acciones solicitadas por el sistema de control centralizado. De esta manera el sistema se ajustaba aún más al funcionamiento de la planificación automática. Esta decisión no sólo fue debida a la complejidad de tener en cuenta estas situaciones, sino también a la precisión del robot ya que en muchos casos era complicado obtener el ángulo preciso de giro, o la distancia recorrida y era difícil volver a recuperar la posición anterior de forma precisa.

Debido a que los robots deben moverse en entorno dinámicos, los cuales van variando según se van realizando las tareas, no es posible poseer toda la información necesaria durante el proceso de planificación, lo cual supone que en muchos casos, si se produce un cambio importante en el entorno, puede que no se pueda llevar a cabo el conjunto de tareas que deben ser realizadas para resolver el problema. Debido a esto y a la naturaleza del planificador utilizado, decidimos dotar al sistema de un proceso de replanificación mediante el cual si aparece una situación en la cual no es posible aplicar la acción enviada por el sistema de razonamiento centralizado, se debería buscar una alternativa, generando un nuevo plan desde el último estado válido hasta la meta.

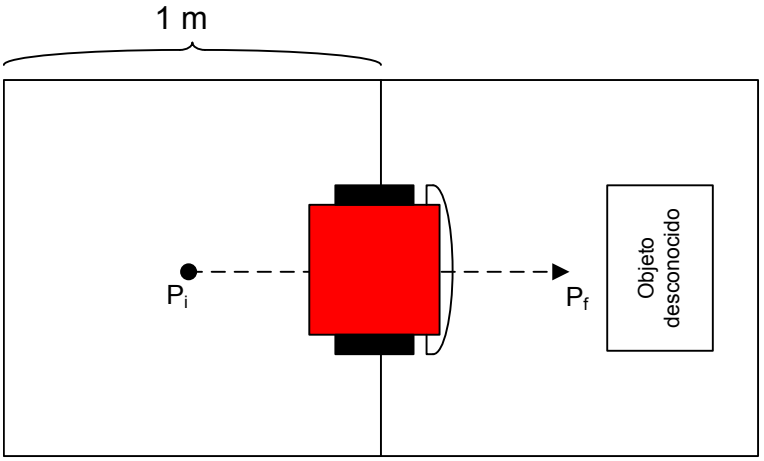


Figura 3.11: Ejemplo de situación de replanificación

Supongamos el ejemplo presentado en la figura 3.11, donde un robot debe moverse desde el punto $P_i$ hasta el punto $P_f$. Cuando ha recorrido la mitad de la distancia entre los dos puntos, es detectado un objeto mediante el sónar del robot. En esta situación el robot podrían realizar dos acciones:

- Replanificar, el robot debería volver a la posición inicial, de forma que deben ser controladas las distancias recorridas por el rover. A continuación modificar la información contenida en el mapa abstracto, indicando que en ese *waypoint* hay un objeto y no es posible pasar por el. Por último volver a ejecutar el planificador desde el anterior estado hasta la meta, con el fin de encontrar un plan válido sin tener que utilizar ese *waypoint*.

- Esperar, debido a que inicialmente ese objeto no debería estar en la posición donde ha sido detectado, el robot podría esperar durante un espacio de tiempo hasta que el objeto continuara su movimiento y dejara de ocupar el *waypoint*. Esto supone que el objeto debe ser móvil, si no lo es el robot deberá esperar por un tiempo infinito.

Ninguna de estas dos posibles acciones, nos ofrece un buena solución, ya que es posible que mediante ninguna de ellas podamos resolver el problema, bien por que el objeto no se mueva nunca o bien por sea obligatorio pasar por ese *waypoint* para resolver el problema. Por lo tanto, teniendo en cuenta situaciones como la presentada, se planteó la posibilidad de aunar ambos comportamientos, debido principalmente a que el proceso de replanificación es complejo y una vez que se decide replanificar esto afecta de manera significativa a la posible solución generada o incluso a la consecución de las metas. De esta forma antes de replanificar es necesario estar seguro de que es necesario realizarlo; por lo tanto en el caso de que un robot detecte un objeto que se interponga en su camino deber esperar un tiempo prudencial hasta decidir replanificar. Incluso seria posible analizar la trayectoria del objeto teniendo en cuenta la información recogida por el sónar, pero esto tampoco nos asegura totalmente que el objeto siga moviéndose si lo hacia previamente o se mueva si inicialmente no lo había realizado.

## 3.6. Flujo general de ejecución

A lo largo de todo este capítulo se ha realizado una descripción detallada de los distintos elementos de la arquitectura y de las técnicas que han sido utilizadas para el control de cada uno de los robots, pero en ningún momento se ha presentado de forma detallada el flujo de control que se produce durante la ejecución del sistema. En este apartado se realiza una descripción explícita de los distintos procesos por los cuales pasa el sistema de control desde que se inicia la ejecución en el controlador central hasta que el proceso finaliza su ejecución. En la figura 3.16 se presenta un diagrama que describe de forma precisa el flujo de ejecución que el sistema de control sigue cuando trata de resolver un problema.

En el diagrama 3.12 se presenta el flujo de ejecución del sistema de control central a la izquierda y el flujo de control de uno de los robot a la derecha, se han incluido ambos flujos de control para representar como afectan los mensajes que se envían y se reciben en el flujo de ejecución del sistema.

- Carga entorno: Esta tarea comienza con la lectura y carga de los parámetros incluidos en el fichero de configuración. A continuación se realiza el análisis de los ficheros de definición del dominio y del problema (para más información ver anexo c) y se construye la representación simbólica. En caso de todos los fichero hayan sido leídos correctamente y no haya producido ningún error en los procesos de análisis se realiza la ejecución del planificador y en caso de obtener solución se pasa a la siguiente tarea.

- Generar Mapa: Mediante esta tarea, se realiza una transformación de la información de alto nivel, que ha sido obtenida tras la lecturas de definición del problema. Para realizar dicha transformación se ejecutan una serie de procesos que obtienen la información de todos los predicados y la transforma a una representación de bajo nivel, dependiente del dominio. Que permite la monitorización del estado del entorno según se van ejecutando las acciones del plan obtenido tras la ejecución del planificador.

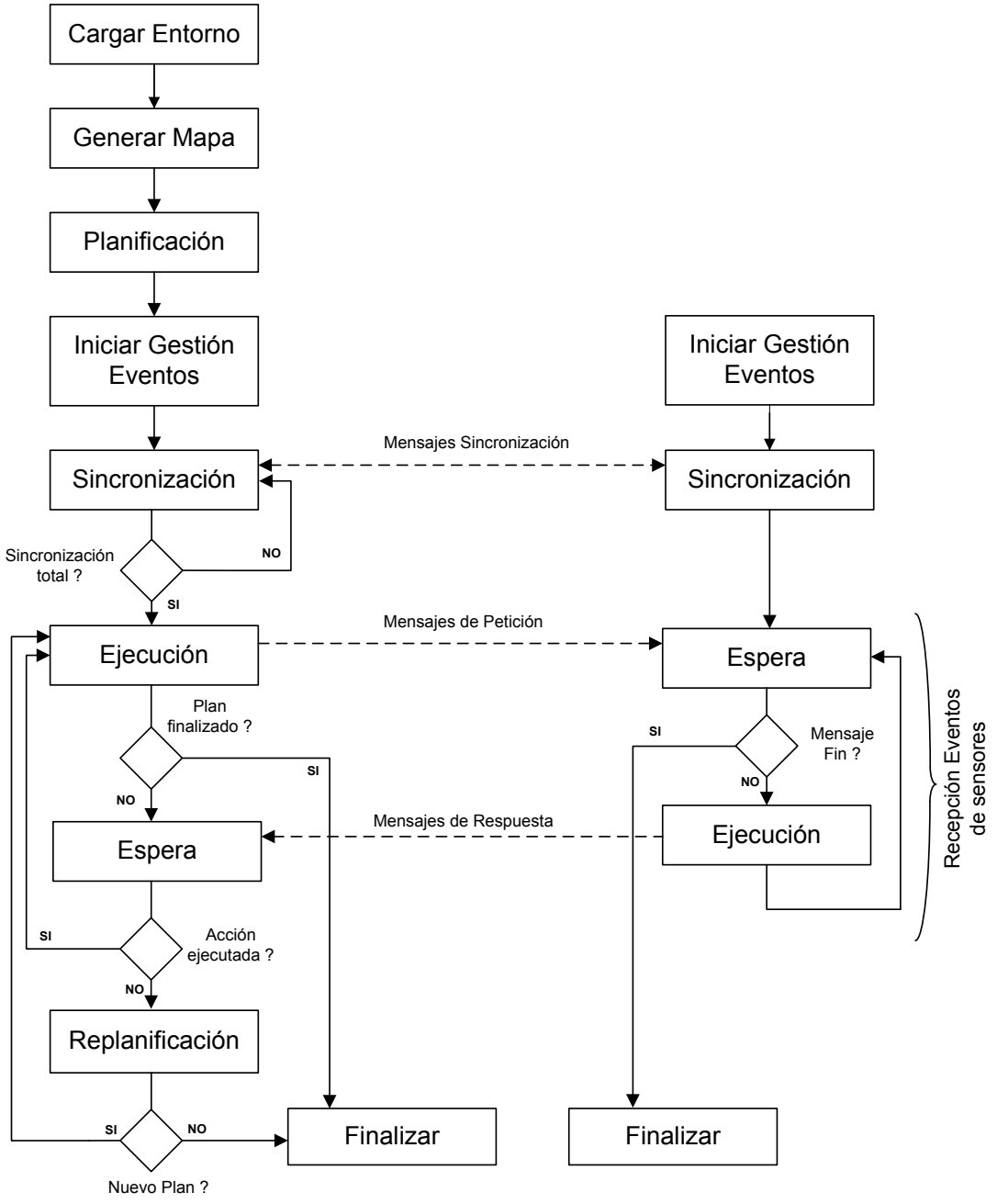


Figura 3.12: Diagrama de flujo global del sistema

- Planificación: Es esta tarea se realiza la ejecución del planificador. Con el fin de obtener el secuencia de acciones que deben ser aplicadas para resolver el problema, en esta fase se realiza la definición de la estructuras de control

para la gestión de las acciones.

- Iniciar Gestión de Eventos: Esta es una tarea que genera las distintas asociaciones entre servicios y asigna los manejadores para los distintos eventos o mensajes que van a ser recibidos.

- Sincronización: Mediante esta tarea el sistema se produce la sincronización entre el sistema de control central y los distintos robots, asignándoles los identificadores definidos en el problema.

- Ejecución: Esta tarea se corresponde con el estado de ejecución de todos los autómatas y se utiliza para la realizar los procesos de selección y envío de las acciones a los robot o espera de nuevas acciones a realizar por parte de los robots.

- Espera: Esta tarea se corresponde con el estado de espera de todos los autómatas y es un estado en los cuales los distintos controladores esperan la llegada de un mensaje. En el caso de los robots para realizar una acción y en el caso del controlador central para conocer el resultado de la acción enviada previamente.

- Replanificación: Esta tarea se corresponde con el proceso de replanificación presentado en el autómata del sistema de control central y que se mostraba en el diagrama 3.5. Durante este proceso se realiza la eliminación de las distintas acciones que ya no pueden ser generadas, se realiza la actualización del mapa abstracto y se realiza una nueva planificación desde el último estado correcto, antes del error. En caso de que se encuentre un nuevo plan el sistema continuará la ejecución normal, en caso contrario al no haber encontrado plan, se finalizará el proceso de ejecución.

- Finalizar: Esta tarea representa el estado final del sistema de control y se consigue cuando el robot consigue resolver el problema planteado o se produce un error que le impide obtener todas la metas. Cuando se llega a este

estado se produce la eliminación de las asociaciones con los demás servicios y se finaliza el proceso de ejecución.

# Capítulo 4

# Experimentación

En esta apartado se realiza una descripción de los experimentos que han sido definidos para comprobar el correcto funcionamiento del sistema de control desarrollado en este proyecto. Para cada uno de ellos se presenta un mapa de los distintos elementos presentes en el entorno, indicándose cuáles son conocidos y cuáles no por el sistema de planificación. A continuación se realiza una descripción de los distintos objetivos y de los problemas con los cuales se ha encontrado el sistema a la hora de resolver el problema. Por último, para cada uno de los robots se presenta una conjunto de datos, referentes a la actividad del robot en el entorno real.

## 4.1. Definición de los experimentos

Para cada uno de los experimentos descritos en este capítulo, se presenta una representación gráfica mediante el cual se describen los distintos elementos que forman parte del entorno. Con el fin de facilitar la compresión de los elementos incluidos en cada uno de los mapas, se realiza una descripción detallada del significado de cada uno de los elementos que podrán aparecer en cada uno de ellos.

Esta imagen representa un objeto en movimiento no conocido inicialmente, la flecha indica la dirección del movimiento del objeto.



Esta imagen representa un muestra de tierra que debe ser recogida por un robot de tipo transportador.

Esta imagen representa un muestra de roca que debe ser recogida por un robot de tipo transportador.

Esta imagen representa a un robot de tipo transportador, el cual puede desplazarse entre los distintos waypoints.

Esta imagen representa a un robot de tipo excavador. La sección de color amarillo representa la zona de carga en la que se debe situar un robot de tipo transportador para que los objetos recogidos por el excavador sean depositados en el depósito de carga.

Esta imagen representa un objeto estático que no era conocido previamente y por tanto el plan obtenido por el planificador puede poseer acciones que no puedan ser ejecutadas.



Esta imagen representa una conexión bidireccional entre dos puntos del mapa (waypoints). Indica que los robots transportadores pueden moverse en ambas direcciones.



Esta imagen representa un conexión unidireccional entre dos puntos del mapa (waypoints). Indica que los robot transportadores sólo pueden moverse en la dirección de la flecha.

## 4.2. Nomenglatura de los resultados

En esta apartado se realiza una descripción del significado de cada uno de los elementos que aparecen en las tablas resumen, que describen el resultados de los experimentos que han sido realizados sobre la arquitectura.

- Número de acciones solución: Mediante este elemento se indica el número total de acciones que han sido ejecutadas para obtener la solución al prob-

lema.

- Tiempo de planificación: Mediante este elemento se indica el tiempo que el planificador **Sayphi** ha tardado en resolver el problema con la información inicial que se conoce al comienzo del proceso de resolución.

- Número de replanificaciones: Mediante este elemento se indica el número de veces que se ha ejecutado el planificador tras el comienzo del proceso de control, entendiéndose como proceso de control, la ejecución de las acciones indicadas tras el primer proceso de planificación.

- Tiempo total de replanificación: Mediante este elemento se presenta el tiempo total de planificación que ha sido empleado en los procesos intermedios de planificación que se han producido durante el proceso de control.

- Número de tiempos de espera: Como se indico en el capitulo 3 de este documento, cuando el sistema detecta un objeto no conocido inicialmente espera un tiempo con el fin de descubrir si el objeto es móvil o no. Mediante este elemento se indica el número de veces que el sistema de control espera a que un objeto se aparte de su trayectoria.

- Tiempo total de espera: Mediante este elemento se indica el tiempo total que el sistema ha esperado por un objeto. El tiempo máximo de espera puede ser configurado en el sistema de control, mediante el fichero de configuración.

- Perdida de precisión: Mediante este elemento se indica el grado de perdida de precisión que se ha producido al realizar el experimento. Los posibles valores son los siguientes:

  - Ninguna: No se ha producido perdida de precisión apreciable.
  - Baja: Se ha producido perdida de precisión, pero esta no ha ha impedido finalizar el plan de acciones.
  - Media: Se ha producido perdida de precisión que han supuesto problemas a la hora de ejecutar el plan de acciones.

- Alta: Se ha producido perdida de precisión graves que han impedido la ejecución correcta del plan.

## 4.3. Descripción de los experimentos

A continuación se presenta la descripción de los 5 experimentos que han sido ejecutados y resueltos mediante el sistema de control desarrollado para este proyecto.

### 4.3.1. Experimento 1

En la figura 4.1 se presenta la descripción gráfica del primer experimento, este está constituido por 6 zonas o *waypoints* de 80 cm, entre las cuales debe desplazarse un robot de tipo transportador para resolver el problema,

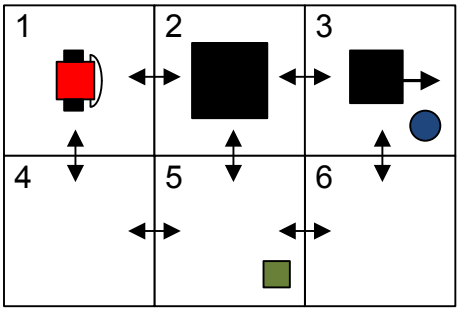


Figura 4.1: Descripción gráfica del experimento 1.

El objetivo de este experimento consiste en que el robot de tipo transportador recoja la muestra de suelo que se encuentra en *waypoint* 5, y la muestra de roca que se encuentra en la campo 3. Una vez recogidas ambas muestras el robot debe volver a su posición original.

Observando la figura 4.1 se puede apreciar que puede surgir un problema durante el proceso de resolución debido a que existe un objeto móvil en el *waypoint* 6, por el cual debe pasar el robot obligatoriamente para conseguir realizar su misión. En los resultados presentados en la tabla 4.2 se observa que el robot

| Resultado del experimento | |
|---|---|
| Número de acciones solución | 10 |
| Tiempo de planificación | 0,063 s |
| Número de replanificaciones | 0 |
| Tiempo total de replanificación | 0.0 s |
| Número de tiempos de espera | 1 |
| Tiempo total de espera | 10 s |
| Perdida de precisión | Ninguna |

Tabla 4.2: Resultados experimento 1

se encontró con un objeto móvil, pero este pasó de largo, antes de que se hubiera consumido el tiempo máximo de espera del robot, permitiendo a este continuar realizando sus acciones, por lo tanto no fue necesario replanificar para poder resolver el problema.

En la url presentada a continuación es posible ver una grabación del experimento 1, realizada en las instalaciones de la Universidad Carlos III de Madrid.

`http://www.buenhombre.com/robots/?p=9`

### 4.3.2. Experimento 2

En la figura 4.2 se presenta la descripción gráfica del segundo experimento, en este caso el entorno está formado por 8 *waypoints* de 80 cm, entre los cuales debe desplazarse un único robot de tipo transportador.

El objetivo de este experimento consiste en que el robot de tipo transportador recoja una muestra de suelo que se encuentra en el *waypoint* número 6 y una muestra de roca en el waypoint número 4. Dependiendo del plan que sea generado por el planificador para este experimento, podría ser necesario que el sistema solicite el proceso de replanificación, debido a que existe un objeto desconocido en una de las rutas que puede tomar el robot transportador cuando vaya a recoger la muestra de roca de la posición número 4.

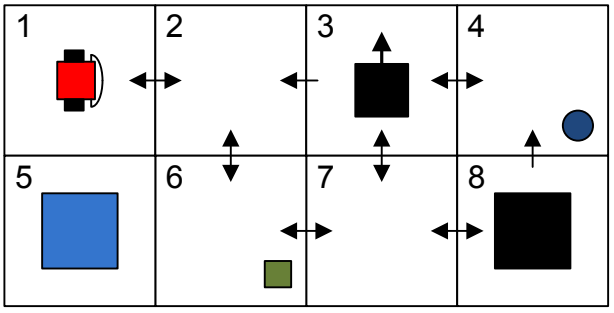


Figura 4.2: Descripción gráfica del experimento 2.

Como se puede observar en los resultados presentados en la tabla 4.3, no ha sido necesario ejecutar el proceso de replanificación, aunque sí ha sido necesario esperar a que el objeto móvil que se encontraba en el punto número 6 se desplazará y permitiese al robot de tipo transportador finalizar su recorrido.

| **Resultado del experimento** | |
|---|---|
| Número de acciones solución | 16 |
| Tiempo de planificación | 1.0 s |
| Número de replanificaciones | 1 |
| Tiempo total de replanificación | 5.4 s |
| Número de tiempos de espera | 0 |
| Tiempo total de espera | 0 s |
| Perdida de precisión | Baja |

Tabla 4.3: Resultados experimento 2

En la url presentada a continuación es posible ver una grabación del experimento 2, realizada en las instalaciones de la Universidad Carlos III de Madrid.

`http://www.buenhombre.com/robots/?p=12`

### 4.3.3. Experimento 3

En la figura 4.3 se presenta la descripción gráfica del tercer experimento descrito en este documento. En este caso el entorno se encuentra constituido por 10 *waypoints* de 80 cm cada uno y una zona de carga de 35 cm. En este experimento se utilizan dos robot, uno de tipo transportador y uno de tipo excavador que se encuentra situado en el *waypoint* 10.

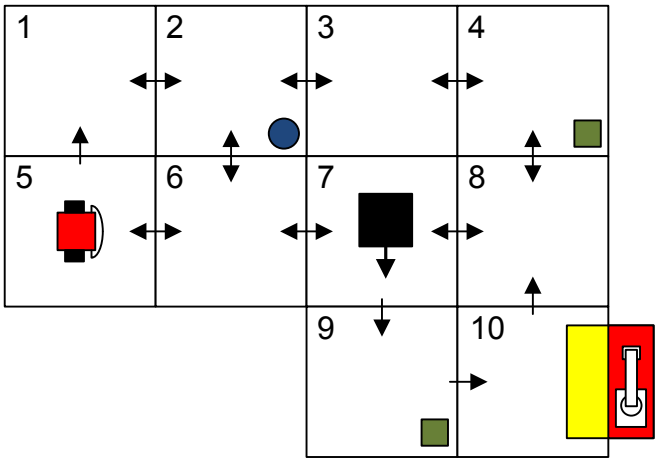


Figura 4.3: Descripción gráfica del experimento 3.

En este experimento el objetivo del robot explorador es recoger las dos muestras de arena y de rocas que se encuentran distribuidas por el entorno. Además, debe desplazarse hasta la zona de carga del robot excavador y esperar a que le deposite un objeto recogido previamente por este, volviendo finalmente al *waypoint* número 1.

Como se puede observar en los resultados presentados en la tabla 4.4, el robot transportador no ha tenido que enfrentarse a ninguna contingencia, ya que no ha sido necesario ni esperar o ejecutar la replanificación para poder resolver el problema propuesto.

En la url presentada a continuación es posible ver una grabación del experimento 3, realizada en las instalaciones de la Universidad Carlos III de Madrid.

`http://www.buenhombre.com/robots/?p=15`

| Resultado del experimento | |
|---|---|
| Número de acciones solución | 20 |
| Tiempo de planificación | 1 s |
| Número de replanificaciones | 0 |
| Tiempo total de replanificación | 0.0 s |
| Número de tiempos de espera | 0 |
| Tiempo total de espera | 0 s |
| Perdida de precisión | Baja |

Tabla 4.4: Resultados experimento 3

### 4.3.4. Experimento 4

La descripción del cuarto experimento se presenta en la figura 4.4, en ella se observa un problema formado por 19 *waypoints* de 80 cm cada uno y una zona de carga de 35 cm, así como dos robots de diferente tipo.

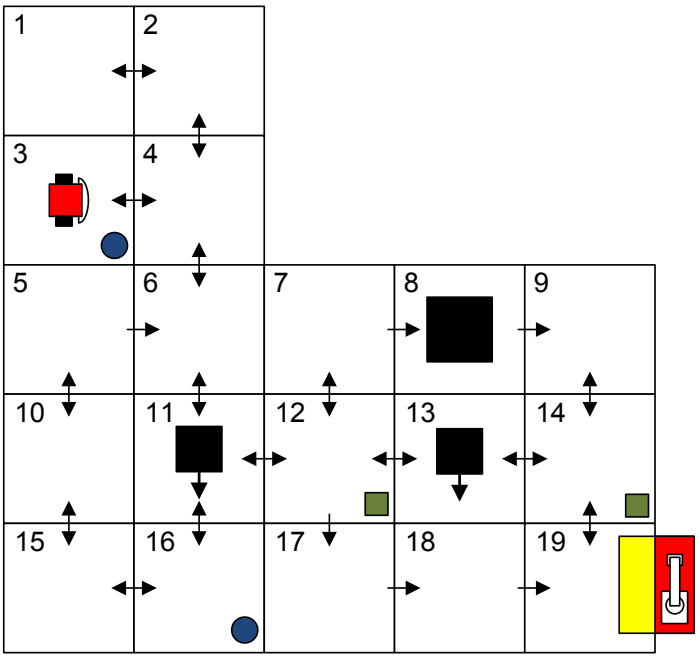


Figura 4.4: Descripción gráfica del experimento 4.

El objetivo de este experimento es similar al anterior, consiste en recoger todas las muestras de arena y rocas que hay en el entorno, así como transportar un objeto que debe ser previamente recogido por el robot excavador. Una vez recogidas todas las muestras el robot transportador debe llevar las muestras recogidas al punto número 1.

| **Resultado del experimento** | |
|---|---|
| Número de acciones solución | 33 |
| Tiempo de planificación | 95.67 s |
| Número de replanificaciones | 0 |
| Tiempo total de replanificación | 0.0 s |
| Número de tiempos de espera | 1 |
| Tiempo total de espera | 10 s |
| Perdida de precisión | Media |

Tabla 4.5: Resultados experimento 4

Este experimento es mucho más complejo que los anteriores, no sólo por el número de objetos desconocidos y/o móviles que existen en el entorno, sino porque debido al incremento en el tamaño del entorno y la complejidad de las metas a conseguir. Esto aumenta el número de errores de precisión que se van acumulando en los robots y por tanto disminuye la precisión en sus movimientos. En este experimento la precisión es muy importante, ya que el robot transportador debe entrar en la zona de carga y descarga del robot excavador con precisión para que este último pueda depositar los objetos en la zona de carga.

En la url presentada a continuación es posible ver una grabación del experimento 4, realizada en las instalaciones de la Universidad Carlos III de Madrid.

`http://www.buenhombre.com/robots/?p=17`

### 4.3.5. Experimento 5

En la figura 4.5 se presenta el quinto experimento realizado sobre el sistema de control. Esta ha sido el experimento más complejo realizado, debido a la longitud de los planes generados por el planificador para la resolución del problema. El problema está formado por 23 *waypoints* de 80 cm cada uno, una zona de carga de 35 cm y dos robots de diferente tipo.

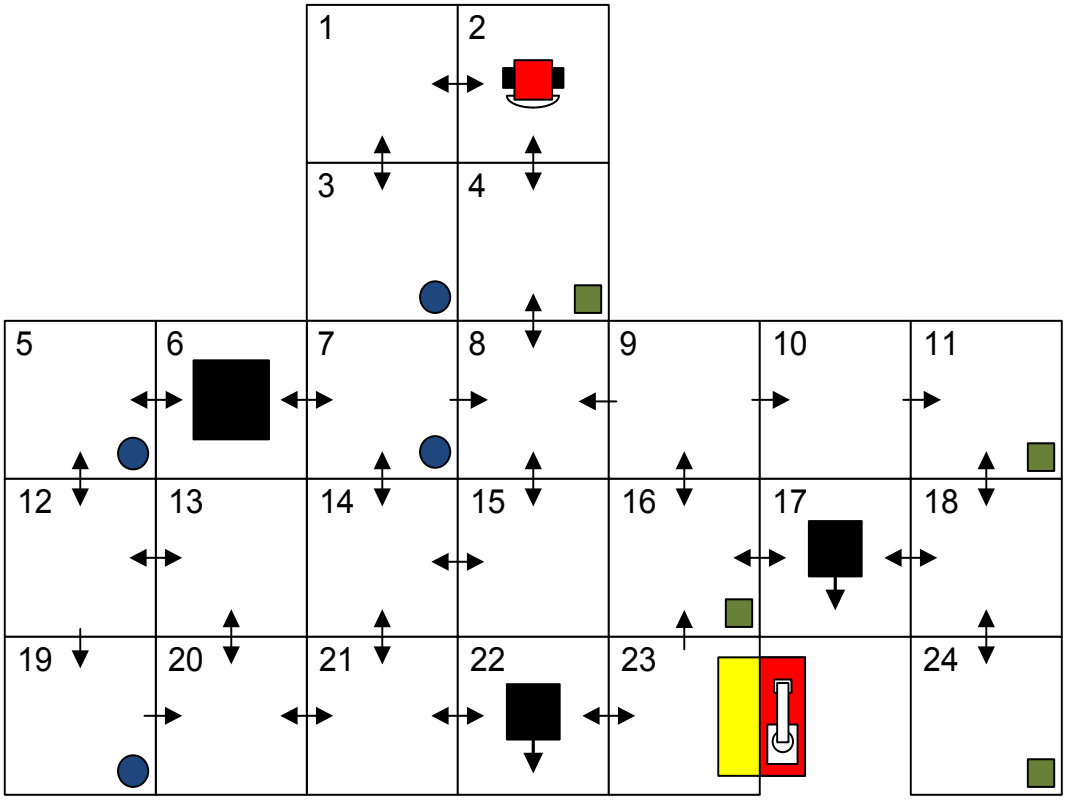


Figura 4.5: Descripción gráfica del experimento 5.

La solución a este problema consiste en la recolección de todas las muestras de arena y rocas, y la obtención de un objeto que ha sido recogido por el robot de tipo excavador, llevándo todos los elementos recogidos a la posición de origen del robot de tipo transportador.

Tras analizar los resultados presentados en la tabla 4.6, observamos que el sistema de control ha tenido bastante problemas a la hora de resolver este

| Resultado del experimento | |
|---|---|
| Número de acciones solución | 58 |
| Tiempo de planificación | 205.86 s |
| Número de replanificaciones | 3 |
| Tiempo total de replanificación | 61.18 s |
| Número de tiempos de espera | 1 |
| Tiempo total de espera | 10 s |
| Perdida de precisión | Alta |

Tabla 4.6: Resultados experimento 5

experimento, ya que sido necesario ejecutar el proceso de replanificación al detectar el objeto estático que se encuentra en el *waypoint* número 6. Además debido a los errores que se van acumulando, el robot P3-DX se ha perdido, esto ha supuesto que se produjeran dos replanificaciones al haber sido detectados los muros que hay en el entorno de pruebas, que se encuentra fuera de los *waypoints*. Esto ha supuesto que no se pudieran satisfacer todas las metas. Cabe destacar que debido a la longitud y complejidad del experimento en todas las ejecuciones que se realizaron, el robot tuvo problemas en el entorno, debido a la precisión en la utilización de los actuadores, que producían pequeños errores los cuales se iban acumulando y han producido que el robot se salieran de los *waypoints* impidiendo la finalización del plan de forma correcta.

En la url presentada a continuación es posible ver una grabación del experimento 5, realizada en las instalaciones de la Universidad Carlos III de Madrid.

`http://www.buenhombre.com/robots/?p=21`

## 4.4. Conclusiones a los experimentos

La realización de experimentos me ha permitido observar el correcto funcionamiento de la arquitectura de control que ha sido desarrollada. Además, me

ha permitido descubrir también algunas de las complicaciones que surgen a la hora de desplegar una arquitectura de control sobre robots reales, en los cuales surgen problemas relacionados con la precisión de los sensores y los actuadores (odometría), que en algunos casos suponen que los robots introduzca pequeños errores en sus trayectorias y acciones. Estos errores junto con la complejidad y longitud del plan de acciones pueden suponer en algunos casos que el robot **se pierda**, es decir se encuentre en una posición distinta a la real.

# Capítulo 5

# Gestión del proyecto

En este capítulo se realizado una descripción de la fases que conforman el proceso de desarrollo de este proyecto, describiendo las tareas realizadas en cada una de ellas. De forma conjunta se incluirá un presupuesto indicando los coste materiales y personales que serían necesario para realizar este proyecto así como un diagrama Gantt donde se presentan de forma gráfica las fases, las tareas que las componen y las dependencias entre las mismas (dependencias de orden de realización).

## 5.1. Distribución de tareas

En la tabla 5.1, se presenta el conjunto de tareas realizadas a lo largo de este proyecto, para cada una de ella, se indican el nombre de la tarea, el número días utilizados para la realización de las tareas, la fecha de comienzo de la tarea y la fecha de finalización. Cabe destacar que esta planificación ha sido realizada de la forma más realista posible, de modo que cada uno de lo días sólo computa como **4 horas laborables**. Además, se han incluido dos períodos de vacaciones, uno comprendido entre 1 de Julio de 2010 y el 31 de Agosto de 2010 y otro desde el día 24 de Diciembre de 2010 hasta el 10 de Enero de 2010.

En la figura 5.1 se presenta el diagrama de Gantt construido para este proyecto. Este ha sido fragmentado en 5 pequeños diagramas para facilitar su

| Tarea | Duración | Comienzo | Fin |
|---|---|---|---|
| **Análisis** | 30 días | 05/10/09 | 13/11/09 |
| Análisis de los dispositivos | 20 días | 05/10/09 | 30/10/09 |
| **Toma de Requisitos** | 10 días | 02/11/09 | 13/11/09 |
| Definición de funcionalidades | 6 días | 02/11/09 | 09/11/09 |
| Definición de restricciones | 4 días | 10/11/09 | 13/11/09 |
| Definición de Casos de Uso | 6 días | 16/11/09 | 23/11/09 |
| **Aprendizaje de la tecnología** | 88 días | 24/11/09 | 25/03/10 |
| Aprendizaje MRDS | 40 días | 24/11/09 | 18/01/10 |
| Desarrollo controladores Prueba | 36 días | 19/01/10 | 09/03/10 |
| Aprendizaje Gold Parser | 4 días | 10/03/10 | 15/03/10 |
| Aprendizaje MindSquald | 8 días | 16/03/10 | 25/03/10 |
| **Diseño** | 30 días | 26/03/10 | 06/05/10 |
| Diseño de estructura del sistema | 10 días | 26/03/10 | 08/04/10 |
| Diseño controladores | 12 días | 09/04/10 | 26/04/10 |
| Diseño de gramáticas PDDL | 8 días | 27/04/10 | 06/05/10 |
| Implementación | 50 días | 07/05/10 | 16/08/10 |
| Montaje Brazo Robótico | 4 días | 07/05/10 | 12/05/10 |
| Desarrollo Sistema Control Central | 24 días | 13/05/10 | 15/06/10 |
| Desarrollo Sistema de Control Rover | 8 días | 16/06/10 | 25/06/10 |
| Desarrollo de Sistema de Control Brazo | 14 días | 28/06/10 | 16/08/10 |
| **Pruebas** | 42 días | 17/08/10 | 13/10/10 |
| Pruebas controladores | 12 días | 17/08/10 | 01/09/10 |
| Pruebas interacción planificador | 8 días | 02/09/10 | 13/09/10 |
| Pruebas aceptación globales | 22 días | 14/09/10 | 13/10/10 |
| **Documentación** | 60 días | 14/10/10 | 21/01/11 |

Tabla 5.1: Definición de tiempos y tareas del procesos de desarrollo del software.

visualización en este documento.

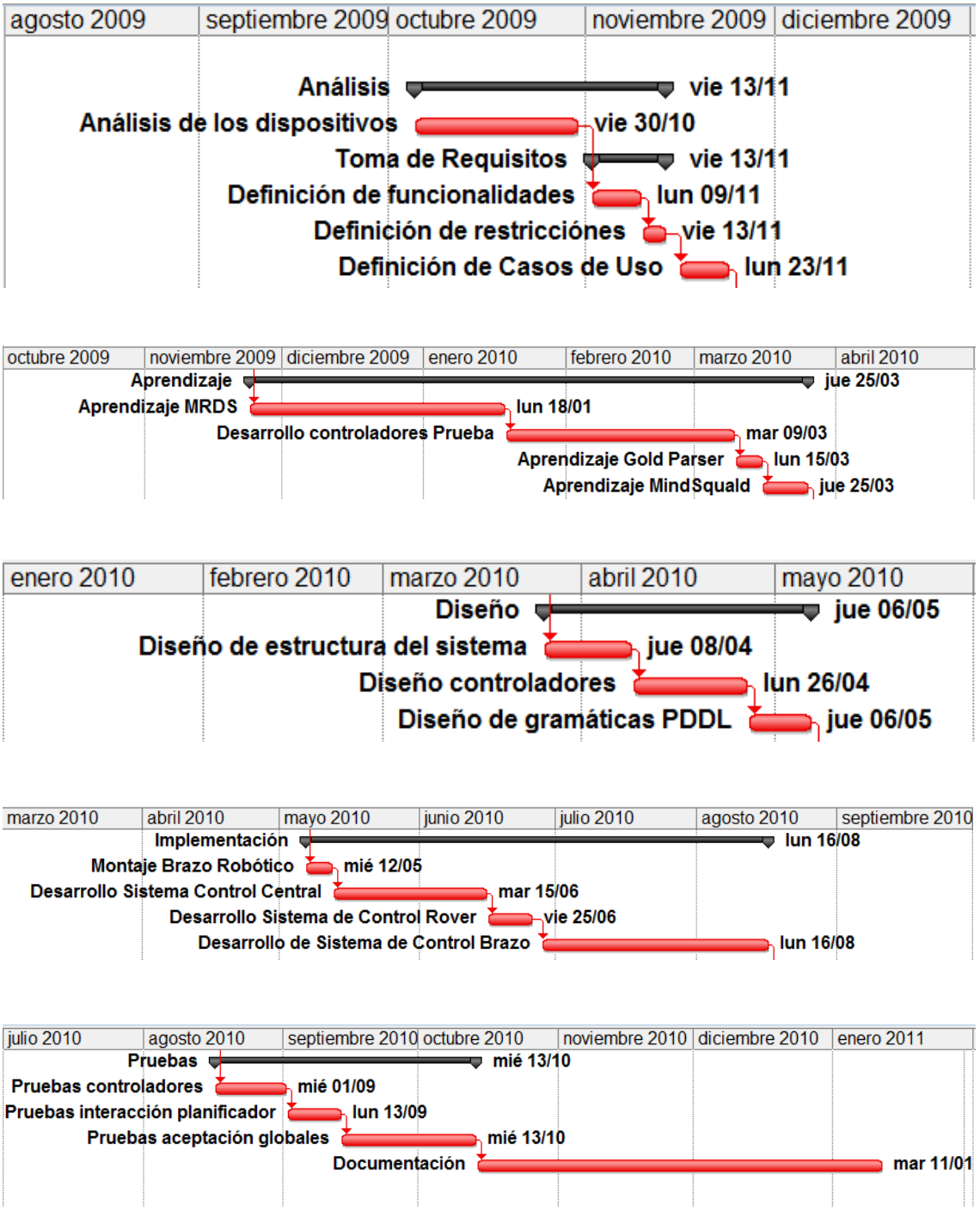


Figura 5.1: Diagrama de Gantt dividido por fases

## 5.2. Descripción de las fases

En primer lugar se presenta un desglose de las distintas fases que contienen cada una de las tareas

### 5.2.1. Análisis

En la **fase de Análisis** se realizó un selección de las líneas generales del proyecto, mediante un toma de requisitos, en la cual se identificaron las distintas funcionalidades que debían realizar los robots, las técnicas que iban a ser utilizadas para producirlas y la tecnología que se tenía disponible para llevarlo a cabo. Como se ha indicado en varias ocasiones en este documento, la tecnología seleccionado para implementar el sistema de control distribuido fue .Net, mediante el lenguaje de programación C#, principalmente debido a que se ha decidido utilizar como *framework* de trabajo Microsoft Robotic Developer Studio 2008 R2, el cual sólo permite la utilización de un pequeño número de lenguajes. Finalmente se decidió implementar un sistema de control híbrido y multi-agente, que utilizará planificación automática, como sistema de razonamiento principal.

### 5.2.2. Aprendizaje de tecnologías

Tras la fase de análisis, se inició la **fase de aprendizaje de tecnologías**. Durante esta fase se adquirieron los conocimientos necesarios para poder llevar a cabo este proyecto. Inicialmente se realizó una toma de contacto con el *framework* de desarrollo seleccionado, lo cual fue muy tedioso, debido a la falta de documentación que existía y al funcionamiento, el cual variaba con respecto a otros *frameworks* de este tipo. Tras conseguir comprender el funcionamiento de los elementos básicos de MRDS, se procedió a la búsqueda de una herramienta de análisis léxico y sintáctico mediante la cual poder procesar los archivos PDDL, que utilizaría el planificador.

Además de todo el proceso de aprendizaje del *framework* fue necesario re-

alizar el montaje del segundo robot, un brazo, y posteriormente encontrar herramientas para poder controlarlo de forma precisa, debido a que las ofrecidas inicialmente por MRDS no funcionaban correctamente. Cabe destacar que esta tal vez fue la fase más larga y tediosa del proyecto debido a los diferentes problemas que fueron surgieron para controlar los distintos robots, así como la falta de documentación referente al funcionamiento a MRDS.

### 5.2.3. Diseño

En la **fase de Diseño** se realizó una descripción de la estructura del sistema de control, los procesos de comunicación entre los distintos dispositivos y los controladores reactivos que gestionarían cada uno de los robots que iban a ser utilizados. Es necesario mencionar que esta fase comenzó de forma paralela a la de aprendizaje, debido a que la estructura básica del sistema de control fue diseñada inicialmente, pero fue refinada teniendo en cuenta las ventajas y desventajas del *framework* de desarrollo seleccionado, del cual no se tenía un alto conocimiento al comienzo del proceso de diseño de este proyecto y según avanzaba la fase de aprendizaje se iban obteniendo conocimientos referentes al modelo de funcionamiento y comunicación, los cuales han influido de forma significativa en la estructura final de la arquitectura de control $AMaC^2$.

Una vez finalizada esta fase, se definío una estructura basada en tres niveles, el primero de ellos gestionaría todo lo referente al proceso de planificación y la gestión de la información obtenida a través de los robots y que fuera necesaria, para poder planificar de nuevo si fuera necesaria. La segunda se encargaría de la gestión de las comunicaciones mediante los sistemas de comunicación ofrecidos por MRDS 2008 R2 y la asignación de tareas entre los distintos robots. Por último la tercera estaría formada por los controladores de tipo reactivo desarrollados para cada uno de los robots seleccionados.

### 5.2.4. Implementación

En esta fase se realizó el desarrollo que había sido obtenido en la fase anterior, el proceso de desarrollo comenzó con el desarrollo del sistema de control para el robot principal el P3-DX, lo cual fue muy costoso debido a que las implementaciones iniciales desarrolladas sobre el simulador que proveía MRDS no se adaptaban fielmente a la realidad. Esto supuso graves problemas y tuvo que ser implementado de nuevo el controlador directamente sobre el robot real.

A continuación se realizó la implementación de las capas deliberativa y de control de la arquitectura. Cuando el desarrollo de ambas capas se encontraba casi finalizado, se detectaron una serie de problemas de funcionamiento en el planificador FF [25], que se producían probablemente debido al sistema operativo en el cual se ejecutaba la arquitectura. Esto supuso la selección de un nuevo planificador Sayphi [26], el cual ya había sido utilizado en otros proyectos en el mismo sistema operativo de forma satisfactoria.

Por último se realizo el desarrollo de la capa reactiva y el montaje del brazo robot. Una vez finalizado el montaje del brazo se detectaron retardos e imprecisiones en los servicios que ofrecía MRDS y se realizo una implementación propia utilizando una librería de control para Lego NXT, denominada MindSquall [29], la cual permitido controlar de manera más precisa el brazo robot.

### 5.2.5. Pruebas

Tras la implementación casi completa del sistema, comenzó la **fase de Pruebas** en un entorno real, utilizando los dos robots. Esta fase fue bastante sencilla excepto por lo continuos problemas que surgieron con el brazo robot, debido a la complejidad que suponía la recogida de objetos de forma precisa. A pesar de todo fue la fase más interesante y divertida, ya que me permitió comprobar el correcto funcionamiento del sistema. Cabe destacar que gracias a la realización de las pruebas iniciales se realizaron algunas modificaciones en el sistema de control y en el dominio de planificación, con el fin de facilitar ciertas tareas que no habían

sido tenidas en cuenta en las fases de análisis y diseño.

### 5.2.6. Documentación

La **fase de Documentación** consistió en el desarrollo de este documento, en el cual se realiza un descripción detallada del sistema de control implementado, las tecnologías utilizadas, los resultados obtenidos, así como toda la información necesaria para poder entender el desarrollo realizado y poder repetirlo.

## 5.3. Presupuesto

En este apartado se realiza una descripción de los costes necesarios para el desarrollo del proyecto descripto en este documento. A continuación se presentan por separado los costes de personal y los costes de los distintos dispositivos y elementos utilizados para el desarrollo del proyecto.

### 5.3.1. Descripción de personal

A continuación en la tabla 5.1, se presentan los costes asociados al personal que ha participado en el desarrollo del presente proyecto, los costes de cada uno de los participantes se han cálculo según las tarifas estipuladas en el BOE 063, de 13/03/2008, sección 3, páginas 15242 a 15243.

| Perfil | Precio hora | Número de horas | Coste final |
|---|---|---|---|
| Jefe de Proyecto | 35 €/hora | 55 | 1.925 € |
| Analista | 33 €/hora | 90 | 2.970 € |
| Investigador | 33 €/hora | 510 | 14.645 € |
| Programador | 29 €/hora | 505 | 14.645 € |
| **Total** | | | 34.185 € |

Tabla 5.2: Costes recursos personales utilizados en el proyecto

Como ya se describío en el apartado 5.1, sólo se han computado 4 horas por día laborale y se han incluido dos períodos de vacaciones, uno comprendido entre

1 de Julio de 2010 y el 31 de Agosto de 2010 y otro desde el día 24 de Diciembre de 2010 hasta el 10 de Enero de 2010.

### 5.3.2. Descripción de los materiales

A continuación en la tabla 5.2, se presentan los costes referentes a los distintos dispositivos y software utilizado, para el desarrollo de este proyecto.

| Recurso | Precio Unidad | Unidades | Precio |
|---|---|---|---|
| Robot P3-DX | 2.954 € | 1 | 99,41 € |
| Cable de conexión Com a usb | 26,95 € | 1 | 26,95 € |
| LEGO Mindstorms NXT v2.0 | 345,81 € | 1 | 345,81 € |
| Conjunto de recursos Mindstorms | 85,95 € | 1 | 85,95 € |
| Rueda de giro Mindstorms | 12,95 € | 1 | 12,95 € |
| Kit de objetos Mindstorms | 12,95 € | 1 | 12,95 € |
| Ordenador portátil, 13 pulgadas | 1.290 € | 1 | 322,5 € |
| Material de oficina | 50 € | 1 | 50 € |
| Impresora Láser Hp 1015 | 125 € | 1 | 125 € |
| Toner Blanco y Negro Hp | 58,95 € | 1 | 58,95 € |
| Microsoft Windows XP Sp3 | 0 € | 1 | 0 € |
| MRDS 2008 R2 | 0 € | 1 | 0 € |
| **Total** | | | 1.130,47 € |

Tabla 5.3: Costes recursos materiales utilizados en el proyecto

Entre los materiales que se presentan en la tabla 5.3, existen algunos elementos coste ha sido calculado teniendo en cuenta el coste de amortización según el tiempo que han sido utilizados durante el proyecto. A continuación se describe para cada uno de estos materiales la forma en la cual se ha calculado su coste.

- Robot P3-DX: Para este material se ha considerado un tiempo de vida de 4 años, y ha sido empleado durante 7 semanas. Siendo el coste utilizado el obtenido mediante la siguiente ecuación.

$$c = 2{,}954 * 7/208 = 99,41D \tag{5.1}$$

- Ordenador portátil: Para este material se ha considerado un tiempo de vida de 4 años, y ha sido empleado durante 1 año. Siendo el coste utilizado el obtenido mediante la siguiente ecuación.

$$c = 1{,}290 * 1/4 = 322,5D \tag{5.2}$$

Tras la realización del computo de la horas empleadas para la realización de este proyecto y el cálculo del coste de los materiales empleados, se ha obtenido que el coste de realización de este proyecto sería de **35.315,47 €**.

# Capítulo 6

# Conclusiones

En este capítulo se realiza una descripción de las conclusiones obtenidas tras las realización de este proyecto, así como el análisis de las dificultades encontradas tras el proceso de diseño e implementación. Al comienzo del capítulo se presentan las conclusiones generales obtenidas tras la finalización del proyecto; a continuación se presentan las conclusiones referentes a los objetivos del proyecto planteados al comienzo del proyecto; y por último se incluyen los problemas detectados a lo largo del proceso de realización de este proyecto.

## 6.1. Conclusiones generales

En primer lugar, este proyecto ha servido para adquirir una serie de conocimientos orientados al diseño y desarrollo de sistemas de control para agente físicos. Además, gracias al desarrollo de este proyecto he comprendido el funcionamiento de los sistemas de control y he podido explorar con más detenimiento la planificación automática, su funcionamiento, los distintos algoritmos que se utilizan, así como sus posibilidades a la hora de ser aplicada en sistema reales como el descrito en este proyecto.

Además el desarrollo de este proyecto me ha permitido observar la complejidad que supone la creación de un sistema de control de este tipo, a pesar de todas las facilidades que me ha ofrecido el *framework* que fue utilizado para el desar-

rollo de este proyecto. Esta complejidad reside principalmente en la cantidad de sistemas que deben ser gestionados y controlados para producir comportamientos que puedan ser considerados inteligentes. Esto implica que un sistema de control debe controlar desde procesos considerados sencillos, como la comunicación entre los distintos elementos que forman el sistema, hasta procesos tan complejos como la interacción que supone el movimiento de un actuador de un robot con los demás elementos de su estructura, o el análisis de la imágenes recogidas por una cámara.

En segundo lugar, el sistema de control desarrollado para este proyecto me ofrece un visión más realista de como un conjunto de robots puede resolver problemas que podemos considerar hasta cierto punto complejos, mediante la utilización de técnicas de la inteligencia artificial, a pesar de que el sistema desarrollado y los experimentos realizados son bastantes sencillos.

En último lugar, este proyecto me ha permitido observar que la planificación es una buena herramienta para definir procesos de actuación complejos de alto nivel, que a su vez se dividen en tareas mucho más sencillas y estas a su vez en tareas más sencillas, que se corresponden con las funciones primarias o básicas de los robots. Este proceso me lleva a la conclusión que tal vez es posible utilizar técnicas de aprendizaje automático que permitan al sistema aprender nuevas acciones, mediante la formada en nuevos elementos del dominio utilizado en el planificador, de forma que sea capaz de enfrentarse a situaciones que no están previstas inicialmente.

## 6.2. Conclusiones referentes a los objetivos

En este apartado se presenta un análisis de las conclusiones sobre el grado de cumplimiento de cada uno de los objetivos planeados al inicio de este proyecto.

- Estudio y análisis de Microsoft Robotic Developer Studio

  Durante el desarrollo de este trabajo se ha conseguido comprender el funcionamiento de las distintas funcionalidades que ofrece este *framework* de trabajo. Este estudio previo me ha permitido realizar un diseño que se adap-

tara a las restricciones y aprovechara las ventajas de este *framework*. De no haberse realizado previamente este estudio el controlador diseñado inicialmente puede que hubiera tenido que ser modificado parcial o incluso de forma completa debido a ciertas normas que deben ser seguidas a la hora de desarrollador un controlador con esta tecnología. Cabe destacar que el tiempo empleado para la compresión del *framework* ha sido excesivo, en parte debido a la falta de documentación o la calidad de esta y al elevado tiempo empleado en la realización de pruebas con los dispositivos reales.

- Diseño e implementación de la arquitectura

  Esta fase fue tal vez la más interesante de todo el proyecto, ya que en ella se realizó el desarrollo de todo el sistema de control, permitiendo trabajar con diversas técnicas para la gestión de los distintos elementos del sistema. En general esta fase ha sido la más productiva y que más me ha aportado personalmente, ya que me ha permitido trabajar con tecnologías que no utilizo frecuentemente y enfrentarme a la complejidad que supone trabajar con agentes físicos, debido a la amplia cantidad de problemas que surgen, referente a procesos como la sincronización, la obtención de información o la aparición de errores en la ejecución de movimientos.

- Experimentación y evaluación del sistema desarrollado

  Tal vez esta ha sido la fase más compleja de todo el proyecto, no por la complejidad de la definición de los experimentos, lo cual en parte ha sido hasta cierto punto sencillo, sino por el trabajo que ha supuesto la preparación del entorno de trabajo, debido a que era necesario realizar una serie de mediciones para definir las posiciones y tamaños de los distintos *waypoints*, así como las posiciones iniciales de los distintos robot, con el fin de que la información que se le aportaba al planificador fuera lo más precisa. A pesar del costoso trabajo que supone la preparación del entorno real de pruebas, la fase de experimentación pudo ser desarrollada de manera correcta y comprobar que el sistema funcionaba de la forma esperada.

- Desarrollo de la documentación

  El desarrollo de la documentación ha sido muy satisfactorio, ya que me ha permitido aprender a manejar **Látex**. Esto supuso que el proceso de desarrollo de la documentación se alargara mucho más de lo esperado inicialmente, pero me ha permitido trabajar con látex de forma exhaustiva consiguiendo obtener un conocimiento, en mi opinión, bastante avanzado sobre esta tecnología.

## 6.3. Problemas encontrados

En este apartado se presentan los distintos problemas que surgieron durante el proceso de diseño y desarrollo de este proyecto. La mayor parte de los problemas destacables, que supusieron complicaciones para el desarrollo, pueden ser agrupados en tres categorías. La primera referente a los problemas relacionados con el *framework* MRDS, la segunda referente a los distintos dispositivos que han sido utilizados y la tercera y última referente a las herramientas de planificación seleccionadas.

### 6.3.1. Microsoft Robotic Studio

La mayor parte de los problemas que fueron surgiendo durante el desarrollo de este proyecto estuvieron relacionados con el *framework* elegido para su desarrollo, muchos de ellos debidos en gran parte a la escasa y deficiente información que existe. A continuación se realiza una descripción de los distintos problemas que surgieron y las soluciones que fueron aplicadas para solventarlos.

- El primer problema a la hora de utilizar esta tecnología estaba relacionado con la localización de los archivos de los servicios que debían ser ejecutados o compilados. MRDS no permite la ejecución de archivos que se encuentren en cualquier ruta del sistema operativo, estos deben encontrarse siempre dentro de su carpeta de instalación. Por lo tanto a la hora de desarrollar cualquier

sistema de control mediante este *framework*, es necesario almacenar todos los archivos referentes al proyecto que esta siendo desarrollado en un carpeta que se encuentre almacenada en su carpeta de instalación del *framework*.

- Otro de los problemas a los cuales tuve que enfrentarme fue el funcionamiento de los servicios previamente desarrollados y que ofrecía la plataforma MRDS 2008 R2. Con respecto al robot P3-DX, sólo estaban desarrollados los sistema de control del motor, la gestión de los *bumpers* y el láser de detección de objetos. Teniendo en cuenta que el robot utilizado en este proyecto no disponía de un láser, sino de un sonar, era necesario la creación de un servicio o la utilización de otro que permitiera obtener la información generada por este sistema de sensores. Para ello se utilizó el servicio desarrollado por el profesor de la universidad Carlos III de Madrid, Raúl Arrabales. También cabe destacar que los servicios desarrollados para la utilización de los robots Lego NXT no funcionaban correctamente. En muchas ocasiones se producían cortes en la comunicación y muchas de las ordenes enviadas al robot o no eran recibidas o se recibían con retraso, por lo que fue necesario buscar una librería de control para este tipo de robot desarrollada en C#, o que pudiera ser utilizada en Windows, encontrado la librería MindSquall, sobre la cual se tuvieron que realizar algunas correcciones pero que nos ofreció un sistema de comunicación estable entre el servicio de control y el robot.

- Las pruebas iniciales del sistema iban a ser realizadas en el entorno simulado que ofrece MRDS. Pero esto no fue posible, debido a la complejidad que suponía la creación de un modelo tridimensional del brazo robot construido en el entorno virtual. Microsoft no ofrece ninguna herramienta que permita el desarrollo de este tipo de modelos tridimensionales, sólo realiza algunas recomendaciones de cuales podrían ser utilizadas para desarrollarlos. Además, el desarrollo de modelos tridimensionales conlleva una elevada dificultad ya que para poder producir las acciones que puede realizar el robot de forma realista, su estructura debe ser construida de forma precisa, lo que

aumentaría el tiempo de desarrollo de este proyecto y no ofrecía una gran ventaja, ya que estaban disponibles los dispositivos reales para la realización de la pruebas y la comprobación del funcionamiento del sistema de control.

### 6.3.2. Dispositivos

Además de los distintos problemas que surgieron con el *framework* de desarrollo elegido para realizar este proyecto, también aparecieron algunos problemas referentes a los distintos dispositivos que fueron utilizados.

- Con respeto al robot P3DX, se detectó un problema de precisión en los giros y los movimientos, probablemente producido por el controlador que utiliza MRDS. Si el número de giros era muy elevado, aparecía un error en los movimiento que producía que la trayectoria del robot no fuese completamente precisa y en algunos de los experimentos de gran tamaño, muchas veces el robot se salía de las zonas de movimiento permitidas, o realizaba algunas acciones de forma imprecisa. El problema de este error, es que dependía del problema a resolver, del conjunto de acciones que hubieran sido seleccionadas por el planificador y de los posibles problemas que pudiera aparecer a lo largo del proceso de resolución.

- Con respecto al robot Lego NXT, surgieron un mayor número de problemas. El primero es referente a la velocidad del giro de los distintos actuadores que posee, cuya velocidad dependía de la carga de la centralita de control, de forma que aunque se indicara que el actuador girara a una velocidad determinada, esta variaba según el porcentaje de energía de las baterías. Además, otro de los problemas que apareció, consistía en la definición de la posición inicial del brazo la cual no era la misma siempre y esto aumentaba la incertidumbre de que el objeto fuera a ser recogido por el brazo y depositado en el compartimiento de carga del robot P3DX. Por último otro de los problemas que surgieron referentes al brazo robot fue la carga máxima y la forma de los objetos que podía recoger, el cual estaba limitado a pequeñas esferas

de poco peso, u objetos de distintas formas y poco tamaño, lo cual supuso también graves problemas a la hora de utilizar distintos tipos de objetos para ser recogidos.

### 6.3.3. Planificador

Además de los problemas comentados anteriormente, algunos de los cuales eran esperados, teniendo en cuenta el propósito del proyecto, surgieron otros no esperados, referentes al planificador seleccionado para la resolución de los problemas. Inicialmente fue seleccionado el planificador Fast Forward [25], para generar los planes de acciones a realizar los distintos robots, pero debido a múltiples problemas que se producían en algunos casos con el planificador, en mi opinión debidos al sistema operativo utilizado para ejecutarlo, fue necesario sustituirlo por el planificador **Sayphi** el cual ha sido desarrollado por el grupo PLG (Planning and Learning Group) de la Universidad Carlos III de Madrid y ya había sido utilizado sobre el sistema operativo Windows de forma satisfactoria. Cabe destacar que parte del código fuente desarrollado para la gestión de la información generada por el planificador tuvo que ser desechado al cambiar el planificador.

## 6.4. Lineas Futuras

El proyecto descrito en este documento, ofrece la posibilidad de incluir un amplio número de mejoras y ampliaciones, debido principalmente a que ha consistido en la creación de una arquitectura, la cual ha sido desarrollada de forma modular y ofrece la posibilidad de incluir nuevos módulos fácilmente. A continuación se describen algunas posibles ampliaciones o mejoras que podrían aplicarse.

- Una de la principales mejoras a realizar sobre el sistema, está directamente relacionada con la odometría, la cual en muchos casos disminuye las capacidades de los robots a la hora de obtener información del entorno. Esto implica que podría incluirse nuevos sensores, sobre cualquiera de los robots

empleados en este proyecto con el fin de mejorar la interacción con el entorno. La opción más prometedora sería tal vez, la creación de un robot híbrido que utilizara los sensores actuales de los rovers P3-DX y los sensores de bajo costes que ofrece la centralita Lego NXT. Esto permitiría la utilización de sensores de posición como el GPS o el compás o incluso la utilización de uno o varios giroscopios sencillos que nos permitirían obtener información referente a la estructura del terreno. Este tipo de sensores nos permitirían dotar al sistema de un mayor grado de reactividad, debido a que seria capaz de obtener información del entorno de mayor calidad y con mayor precisión. Otra posible opción seria la de equipar al propio robot P3DX con estos sensores, pero esto supondría algunos problemas, debido a que muchos de estos sensores no poseen un controlador de bajo nivel en MRDS 2008 R2 y debería ser desarrollados. Además su coste sería mucho mayor.

- También sería muy interesante la utilización de múltiples robots P3-DX y/o LEGO NXT, con el fin de comprobar si el grado de escalabilidad que teóricamente ofrece MRDS 2008 R2 es real. De esta manera se podrían resolver problemas más complejos y ahondar más en los procesos colaborativos que pudieran producirse en los distintos robots.

- En el caso de utilizar múltiples robots P3-DX, seria posible utilizar un nuevo tipo de robot, denominados exploradores, los cuales utilizarían los controladores de visión desarrollados en trabajo dirigido desarrollado antes de la realización de este proyecto, a los cuales deberían aplicarse algunas modificaciones con el fin de que pudieran reconocer distintos objetos y no sólo robots P3-DX.

- Con respecto al proceso de planificación, seria muy interesante dotar de un mayor realismo al sistema, como por ejemplo mediante la utilización de distancias diferentes entre los distintos *waypoints*, o incluir distintos tipos de terreno de forma que el planificador indicara la velocidad mediante la cual se tiene que realizar la transición entre dos puntos del entorno.

- Debido a que estamos utilizando múltiples robots, los cuales en muchos casos realizan acciones paralelas, sería interesante la utilización de un planificador que tuviera en cuenta esta situación, o crear un módulo de gestión de acciones que realizara una serie de análisis de forma que distribuyera las acciones de forma paralela siempre y cuando no existan restricciones entre ellas. Ya que el sistema de control desarrollado gestiona la representación simbólica utilizada por el planificar sería sencillo construir un módulo de este tipo.

- Otro elemento muy interesante sería la creación de módulo de aprendizaje automático de forma que el sistema de control pudiera aprender conjuntos de tareas, o los propios robots pudieran saber como responder ante ciertas situaciones sin necesidades de solicitar replanificación, simplemente aplicando una serie de operaciones aprendidas previamente que supusieran la modificación del dominio sin necesidad de aplicar el proceso de planificación, en este caso seria necesario que el sistema asegurara la consistencia de las acciones modificadas por el módulo de aprendizaje.

# Apéndice A

# Tecnología

En este apartado se describen los distintos *frameworks*, librerías y aplicaciones que han sido utilizados para el desarrollo de este proyecto.

## A.1. Microsoft Robotic Developer Studio

Microsoft Robotic Developer Studio [28] [27] es un *framework* de desarrollo para el diseño y construcción de sistemas de control para robots, sus principales características son las siguientes:

- Ofrece un sistema de generación de entornos simulados en tres dimensiones mediante el motor AGEIA PhysX, que permite la generación de entornos realistas con un alto grado de precisión e incluso permite la utilización de sensores de captura de imágenes que aumentan el grado de realismo del entorno.

- Permite el desarrollo de arquitecturas orientadas a servicios y eventos, total o parcialmente descentralizadas mediante la utilización de módulos de programación independientes denominados servicios de software descentralizados (Decentralized Software Services), los cuales ofrecen la posibilidad de crear controladores reutilizables para dispositivos concretos como el sónar, los *bumpers*, las cámaras.

- Su sistema de funcionamiento esta construido para permitir la ejecución paralela de procesos, a través de un conjunto de funciones que permite la gestión de distintos hilos de ejecución, lo cual ofrece una gran ventaja a la hora de crear sistemas de control orientados a eventos en tiempo real, ya que permite la obtención de información mediante la ejecución de múltiples proceso de forma concurrente, aunque el grado de concurrencia depende del ordenador que sea utilizado.

- Ofrece la posibilidad de comunicarse con los distintos dispositivos mediante diferentes interfaces de comunicación.

    - Conexión directa mediante *usb*.
    - Conexión inalámbrica mediante tecnología *wireless* o *bluetooth*.

### A.1.1. *Descentralized Software Services*

Este *framework* de trabajo está basado en un modelo orientado a servicios, con el fin de minimizar los costes computacionales de forma que un sistema pueda estar controlado por múltiples servicios. El funcionamiento de estos servicios está basado en un modelo de mensaje-estado, es decir el sistema va variando su estado interno, teniendo en cuenta los mensajes que le llegan debido a las notificaciones emitidas por el resto de servicios. Para poder facilitar una mejor interacción entre los distintos servicios, el sistema se adapta a la arquitectura hardware en la cual se ejecuta, mediante la utilización de grupos de *threads* que permiten la ejecución de múltiples servicios de manera concurrente.

Para poder construir este tipo de sistemas, Microsoft Robotic Developer Studio 2008 R2, ofrece una estructura de programación básica, denominada **nodo DSS** (Decentralized Software Services), la cual nos permite definir un conjunto de operaciones que interactúan de forma conjunta entre sí en un servicio. Además, también ofrece una interfaz de comunicación mediante puertos, para la gestión ordenada y concurrente de los distintos mensajes; así como un sistema de transferencia de información que permite la comunicación con otros servicios. En la

figura A.1, se muestran la estructura básica y los distintos componentes de un nodo DSS.

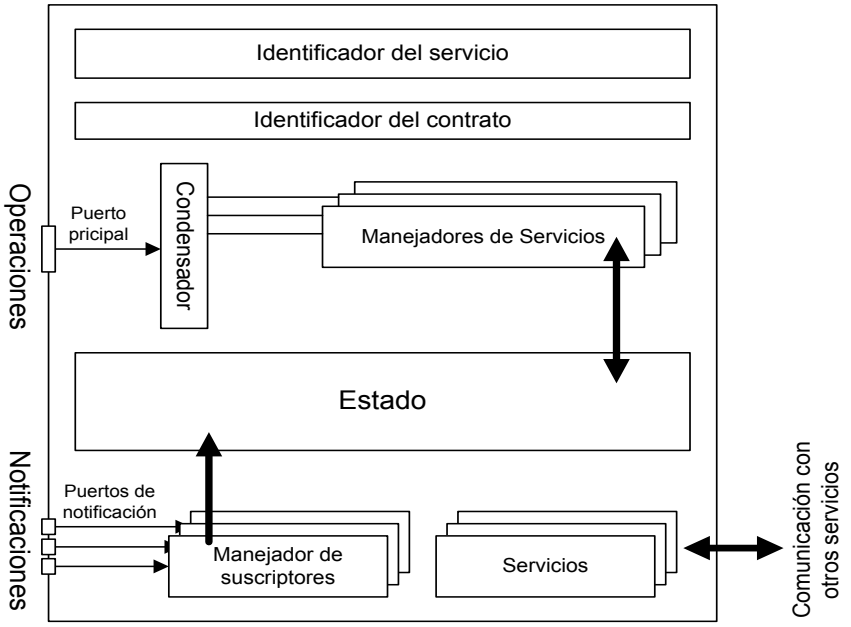


Figura A.1: Estructura de un DSS

- Identificador del servicio: Permite identificar al servicio de forma única en el sistema, su funcionalidad reside en proveer un modo de identificación unívoca con el fin de permitir las comunicaciones y asociaciones con otros servicios.

- Identificador del contrato: Permite describir el comportamiento del servicio, es decir, ofrece una descripción de las distintas operaciones que puede realizar el servicio. También ofrece una descripción de los distintos datos necesarios para ejecutar cada una de las operaciones.

- Estado: Representa el estado del servicio. Se encuentra formado por información referente al servicio, que es accesible por cualquier otro servicio con el cual se haya creado una asociación.

- Socios: Como se describía anteriormente, los servicios son capaces de interactuar con otros servicios mediante la creación de asociaciones, estas asociaciones se generan al inicio del proceso de arranque y permiten acceder a las distintas operaciones ofrecidas por los servicios, así como recibir información procedente de eventos que se han producido en los socios mediante mensajes de tipo notificación. Cabe destacar que las asociaciones son bidireccionales ya que permiten el envío y recepción de mensajes por cualquiera de los dos miembros de la asociación.

- Puertos: Los puertos son el sistema de comunicación básico utilizado por los servicios para el envío y recepción de mensajes, referentes a eventos que se producen en los servicios. Los servicios, poseen dos tipos de puertos. El principal, el cual es el utilizado por el servicio para la emisión de mensajes de tipo interno, y los puertos de notificación que son los utilizados para crear asociaciones con otros servicios, permitiendo la comunicación con otros servicios.

- Manejadores: Son un conjunto de funciones que gestionan los eventos internos y externos, que son producidos por la llegada de un mensaje o notificación al sistema, estas funciones sólo se ejecutan cuando llega una notificación desde el puerto al cual se ha asociado el manejador. En la figura A.3, del siguiente apartado se muestra cómo se produce la asociación entre las funciones o métodos manejadores, los puertos y los tipos de notificación.

- Notificaciones: Las notificaciones son aquellos mensajes que son recibidos a través de los distintos puertos y que se producen cuando se ha generado un evento en un servicio. Por ejemplo cuando un objeto golpea uno de los *bumpers*, se producirá la recepción de un mensaje, que ha sido generado por un evento que indica que ha sido golpeado uno de los *bumpers*, este mensaje normalmente desencadenará la ejecución de un manejador.

### A.1.2. *Concurrency and Coordination Runtime*

El modelo de funcionamiento que propone Microsoft Robotic Developer Studio 2008 R2, permite la utilización simultanea de múltiples servicios, lo que implica que la ejecución y las interacciones que se producen entre ellos deben ser gestionadas de manera ordenada, con el fin de asegurar el correcto funcionamiento del sistema. Teniendo en cuenta estas características, es necesaria la utilización de ***Threads*** (hilos de ejecución) y por tanto es obligatorio contar con una serie de mecanismos de coordinación y sincronización entre los distintos *threads*, que ejecuten cada una de las tareas generadas por los distintos servicios que compiten por los recursos del sistema.

Para facilitar este trabajo el *framework* nos ofrece el sistema CCR (Concurrency and Coordination Runtime), una librería de programación asíncrona que nos ofrece un conjunto de mecanismos que se encargan de la gestión de los procesos de concurrencia y coordinación entre los distintos hilos (*Threads*) de ejecución de procesos. Este sistema permite utilizar la programación *multi-thread* de forma transparente, sin la necesidad que escribir complejos código de ejecución para ofrecer este tipo de mecanismos. A continuación se realiza una descripción de los distintos elementos que ofrece el *framework* para permitir este tipo de mecanismo. En la figura A.2, se presenta un diagrama que presenta a los distintos elementos que forman parte en el proceso de concurrencia y coordinación.

En la figura A.2 se presenta el proceso de coordinación y sincronización de las distintas tareas que pueden ser ejecutadas y que son desencadenadas mediante la recepción de mensajes. A continuación se realiza una descripción de los distintos elementos y/o mecanismos que ofrece el sistema CCR, para la coordinación y sincronización de procesos.

- Puerto de comunicación: El primero de los elementos del sistema CCR, es el puerto de comunicación, este es el mecanismo básico de comunicación que poseen los servicios de MRDS. Su funcionamiento consiste en una cola de tipo FIFO (First-in, First-out), en la cual se almacena los mensajes que son

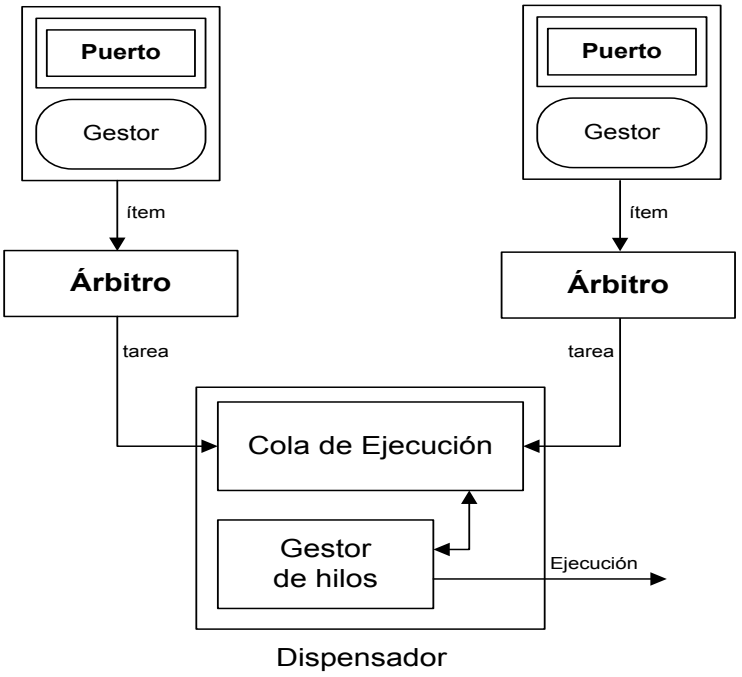


Figura A.2: Sistema de coordinación y sincronización

recibidos y enviados para ser procesadores, como se puede ver el figura A.2. Todo puerto posee un gestor que se ocupa de el tratamiento de los mensajes recibidos y enviados a través de él.

- Árbitros: Como se describió anteriormente todo nodo DSS posee un conjunto de manejadores que se ocupan de ejecutar fragmentos de código fuente tras la recepción de un mensaje. Para poder permitir este proceso de manera concurrente, MRDS nos ofrece un mecanismo de gestión y sincronización denominado árbitro, que permite relacionar un determinado mensaje que debe ser recibido por un puerto específico con un manejador. Además, permite determinar la importancia de los mensajes que van a llegar, indicando el tipo de tratamiento que se les dará. Por ejemplo mediante un árbitro podemos indicar que cuando sea recibido un mensaje de un tipo, el manejador asociado se ejecute de forma exclusiva. En la figura A.3, se muestra uno de los árbitros definidos en el nodo DSS desarrollado para el robot P3-DX, en el se han definido cuatro mensajes, cuyo tratamiento se realiza de manera

concurrente.

```
Activate(
Arbiter.Interleave(
new TeardownReceiverGroup(),
new ExclusiveReceiverGroup(),
new ConcurrentReceiverGroup(
        Arbiter.Receive<control.PlannerRequest>(true,
                                                plannerNotifyPort,
                                                recieveRequestPlanner),
        Arbiter.Receive<drive.Update>(true,
                                        driveNotifyPort,
                                        DriveUpdateNotification),
        Arbiter.Receive<bumper.Replace>(true,
                                        bumperNotifyPort,
                                        BumperReplaceNotification),
        Arbiter.Receive<sonar.Replace>(true,
                                        sonarNotifyPort,
                                        SonarReplaceNotification)
        )
)
);
```

Figura A.3: Ejemplo de definición de árbitro del sistema CCR

- Cola de ejecución: La posibilidad de ejecutar proceso de manera concurrente conlleva algunos problemas, como el acceso a recursos compartidos, o la competición que se producen entre las distintas tareas por un número limitados de núcleos en los cuales ejecutarse. Esto produce que en muchos casos se generen más tareas de las que el sistema puede procesar, teniendo que ser almacenadas en algún tipo de estructura que permita ejecutarlas posteriormente. MRDS ofrece colas de ejecución mediante la estructura **iTask**, que permite almacenar las tareas hasta que sean seleccionadas para ser ejecutadas.

- Dispensadores: El último elemento del sistema CCR, y tal vez el de mayor importancia es el **dispensador de hilos**. Este se encuentra formado por un conjunto o piscina de hilos que son utilizados para la ejecución de las tareas pendientes. A groso modo su función es la de planificar la asignación de las tareas almacenadas en la cola de ejecución a los distintos hilos de ejecución,

teniendo en cuenta ciertas características de las tareas y el número de hilos disponibles.

### A.1.3. Interacciones entre servicios

Para configurar las interacciones entre los distintos servicios descentralizados es necesario definir un fichero XML, este fichero denominado ***manifest*** define el orden de inicio de los distintos servicios, las conexiones y asociaciones que se producen entre ellos.

```
<?xml version="1.0"?>
<!--This file was created with the Microsoft Visual Programming Language.-->
<Manifest xmlns:roverrobot="http://schemas.tempuri.org/2010/09/roverrobot.html"
          xmlns="http://schemas.microsoft.com/xw/2004/10/manifest.html">
  <CreateServiceList>
    <ServiceRecordType>
      <dssp:Contract>http://schemas.tempuri.org/2010/08/planner.html</dssp:Contract>
      <dssp:PartnerList />
      <Name>this:planner</Name>
    </ServiceRecordType>
    <ServiceRecordType>
      <dssp:Contract>http://schemas.tempuri.org/2010/09/roverrobot.html</dssp:Contract>
      <dssp:PartnerList>
        <dssp:Partner>
          <dssp:Contract>http://schemas.tempuri.org/2010/08/planner.html</dssp:Contract>
          <dssp:PartnerList />
          <dssp:Name>roverrobot:planner</dssp:Name>
          <dssp:ServiceName>this:planner</dssp:ServiceName>
        </dssp:Partner>
      </dssp:PartnerList>
      <Name>this:roverRobot</Name>
    </ServiceRecordType>
  </CreateServiceList>
</Manifest>
```

Figura A.4: Ejemplo de fichero manifest

En la figura A.4 se presenta un fragmento de un fichero *manifest*, en el se pueden observar la definición de dos servicios, uno denominado **planner** y otro denominado **roverRobot**. Como se puede observar el segundo se encuentra asociado con el primero, esto quiere decir que el segundo servicio solicitará al primero un enlace de comunicaciones de forma que puedan interactuar entre ellos, mediante el envío y recepción de mensajes.

Con el fin de facilitar la construcción de este tipo de ficheros, Microsoft

incluye un herramienta visual, denominada DSS *manifest* editor, que permite crear de forma sencilla este tipo de ficheros. En la figura A.5, se presenta una captura de pantalla de la herramienta, en ella se muestra el fichero de descripción desarrollado para este proyecto.

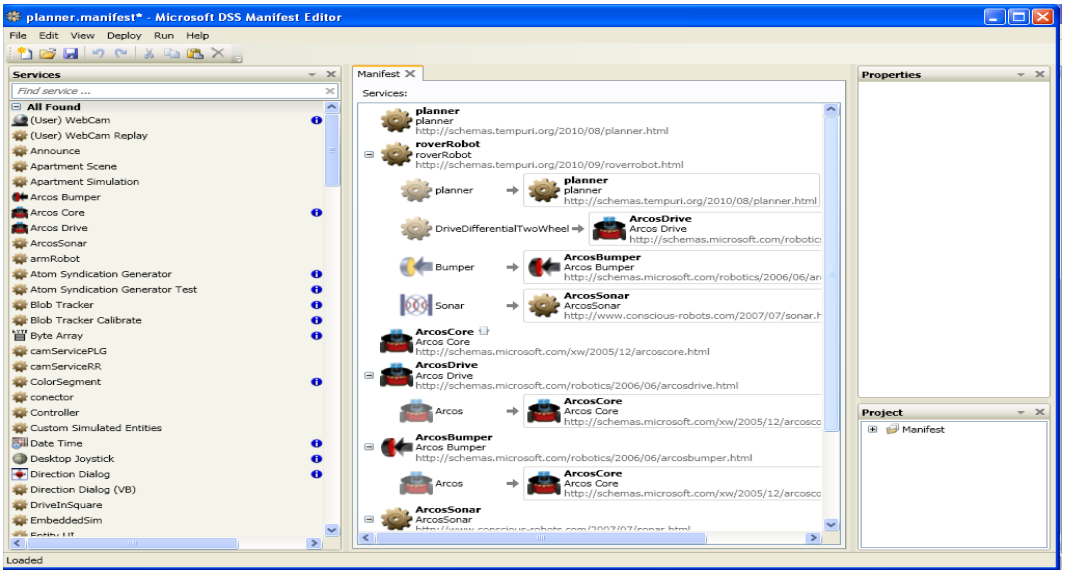


Figura A.5: Captura de la aplicación DSS manifest editor

Como se puede observar, en la figura A.5, el fichero presenta los distintos servicios que han sido desarrollados y utilizados para el sistema de control descrito en este documento, el primero de los servicios que aparece es el referente al sistema de control central, denominado planner, y a continuación se encuentran dispuestos el resto de servicios del sistema. La colocación de los servicios influye en gran medida, ya que el orden en el cual se definen en el fichero indica el orden en el cual se arrancarán los distintos servicios incluidos en el fichero.

### A.1.4. Comandos básicos

En esta apartado se realiza la descripción de una serie de comando básicos, los cuales han sido necesarios para la realización de este proyecto. Estos comandos han sido utilizados para la creación, migración y ejecución de los servicios de software descentralizados que han sido desarrollados. Para la ejecución de estos

comandos es necesario utilizar el DSS Command Prompt, el cual es instalado junto a MRDS.

- Creación: El proceso de creación de los nodo DSS, debe realizarse mediante la ejecución de un comando específico, ya que sino se crean de esta manera no funcionarán de manera correcta ya que no contendrán ciertos elementos de configuración necesarios. El comando para el creación de un servicio, es el presentado en la figura A.6, para el cual al menos hay que especificar el nombre del servicio y el *namespace*, tras la ejecución el nuevo servicio será creado en la carpeta desde la cual se ejecuto el comando.

**DssNewService** /namespace:value1 /service:value2

Figura A.6: Comando de creación de servicios

- Migración: Uno de los principales problemas que surgen al trabajar con las funcionalidades que ofrece MRDS, es que estos no pueden ser portados de una máquina a otra simplemente moviendo el código fuente; e incluso no pueden ser portados de una versión a otra del propio MRDS. Esto implica que es necesario realizar una serie de modificaciones en ciertos ficheros de configuración internos que se crean junto con los servicios, es posible realizar estas modificaciones a mano, pero pueden resultar muy tediosas.

**DssProjectMigration** value1

Figura A.7: Comando de migración de servicios

Para facilitar la migración a otros sistema o a nuevas versiones de MRDS existe un comando específico, presentado en la figura A.7, que permite migrar de forma conjunto entre 1 y n servicios, indicando como parámetro (value1) el directorio que contiene a todos los servicios que desean ser migrados.

- Ejecución: El proceso de ejecución de un servicio o de un conjunto de servicios,difiere con respecto a la ejecución estándar de un programa, para poder ejecutarlos es necesario utilizar un comando que utiliza como parámetros el puerto de ejecución (value1) y el nombre del fichero *manifest* (value2). En la figura A.8 se presenta la sintaxis del comando.

**DssHost** /port:value1 /manifest:value2

Figura A.8: Comando de ejecución de servicios

Además de los tres comandos descritos en este apartado, existen muchos más, cuya sintaxis puede ser consultada a través del DSS Command Prompt.

## A.2. Gold Parser Library

**Gold Parser** [30], es una librería de programación que permite la creación de analizadores de lenguajes. El funcionamiento de este sistema de análisis se basa en la utilización de tres elementos:

- Un fichero de estructuras, en el cual se define las reglas a nivel léxico y sintáctico del lenguaje que debe ser analizado. En este fichero, de tipo **gtc**, se describe la estructura léxica y sintáctica de la gramática. En la figura A.9, se presenta un captura de pantalla del sistema de compilación de las gramáticas, en ella se puede observar un fragmento parcial de una de las gramáticas desarrolladas para este proyecto.

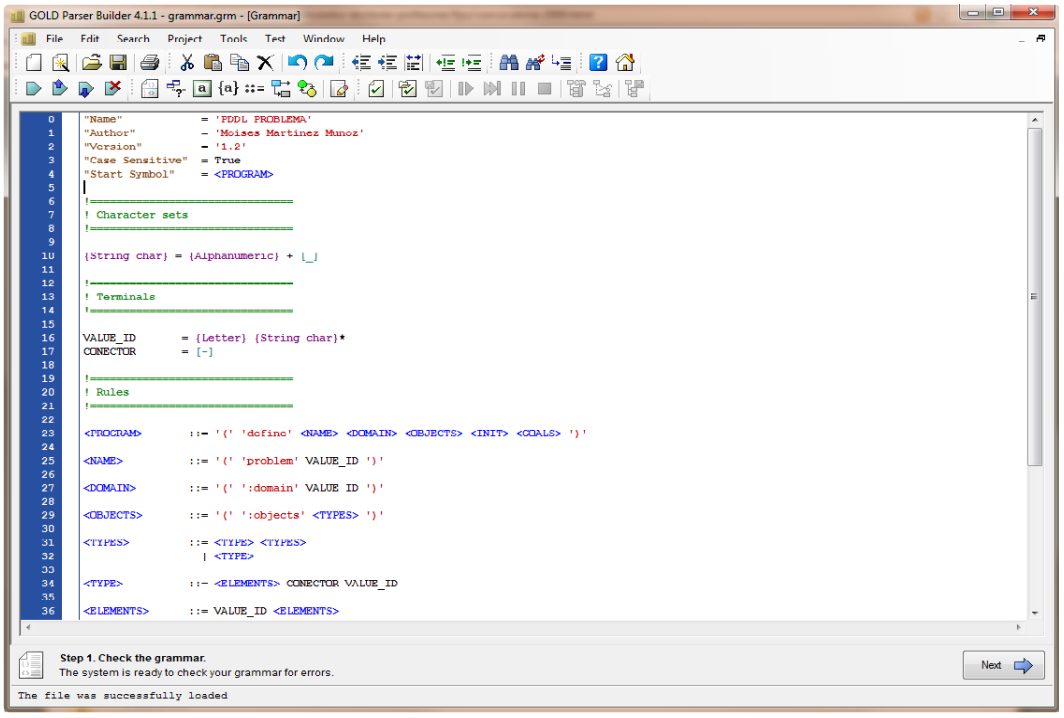


Figura A.9: Captura de la aplicación GoldParser

- Un fichero compilado a partir de la gramática que define el lenguaje, en el cual se encuentran definidas las distintas tablas de transiciones que utilizará el algoritmo LALR(1) (LookAhead LR(1) parsing) para el análisis sintáctico y el autómata finito determinista que será utilizado para identificar las unidades léxicas básicas del lenguaje.

- Un fichero de ejecución, el cual será codificado en un lenguaje de programación específico, en este caso el lenguaje elegido ha sido C#. En este fichero se define el proceso de carga del fichero compilado y el autómata de procesamiento de los ejemplos que quieran ser analizados. Además, en este fichero se incluyen todas las funciones referentes al análisis semántico, creación de listas de tokens, gestión de errores etc.

La decisión de utilizar este tipo de sistema de generación de analizadores de lenguajes, fue en parte debido a que no era posible integrar de manera sencilla con el sistema que se estaba construyendo las aplicaciones Flex y Bison, las cuales

ya había utilizado previamente en otros desarrollos. Además este sistema obliga a crear un fichero independiente con la definición de la gramática, en el cual sólo incluye la estructura de la gramática en BNF y la definición de los distintos tokens. Esto es muy útil a la hora de añadir nuevos elementos a la gramática ya que simplifica enormemente el proceso de modificación debido a que no incluye el código de análisis semántico como ocurre en otros sistema como, por ejemplo, Bison.

## A.3. MindSqualls Library

Debido a los distintos problemas que fueron produciéndose durante la utilización de los servicios ofrecidos por MRDS, para la interacción con los robots de tipo Lego NXT, fue necesario la búsqueda de otro sistema de conexión para interactuar con este tipo de dispositivos. Se seleccionó la librería **MindSqualls** [29]. En la figura A.10, se presenta una fragmento de código de la estructura de las funciones básicas ofrecidas por la librería.

Esta librería ha sido desarrollada en .Net 2.0 y permite el control de forma precisa de la mayor parte de los dispositivos que ofrece la plataforma Lego MindStorm. A pesar de que esta librería supuso muchos ventajas, fue necesario realizar algunas modificaciones sobre el código fuente para optimizar algunos de los procesos que tenía implementados. Por ejemplo, para facilitar la definición de la dirección del giro de los actuadores, o la forma en la cual se envían los bits a través del sistema de comunicación basado en *bluetooth*.

```
// Create a NXT brick on COM40.
NxtBrick brick = new NxtBrick(40);

// Create a motor.
NxtMotor motor = new NxtMotor();

// Attach it to port B of the NXT brick.
brick.MotorB = motor;

// Connect to the NXT.
brick.Connect();

// Run it at 75% power, for a 360 degree run.
motor.Run(75, 360);

// Disconnect from the NXT.
brick.Disconnect();
```

Figura A.10: Fragmento de código fuente de MindSquall

# Apéndice B

# Análisis del sistema

En este anexo del proyecto se realiza una descripción detallada de las funcionalidades y restricciones del sistema de control presentado en este documento, con el fin de describir de manera más técnica el sistema que ha sido desarrollado.

En este apartado no se realiza un análisis exhaustivo del sistema, debido a la extensión que podría suponer para un proyecto de esta envergadura, por lo tanto se ha realiza un análisis sintetizado, que permite exponer de forma técnica los distintos requisitos del sistema. Para ello se presenta una descripción de las características funcionales del sistema, las restricciones que posee a nivel software y hardware, la definición del entorno operacional y finalmente un diagrama de casos de uso, junto a las descripciones textuales de cada uno de los casos de uso.

## B.1. Descripción de las características funcionales

En el tercer capítulo de este documento, se realiza una descripción detallada del funcionamiento de la arquitectura de control que ha sido desarrollada, desde un punto de vista arquitectónico. En este apartado se aporta una visión más detallada desde el punto de visto del desarrollo del software, realizándose una descripción global de las distintas funcionalidades que debía ofrecer el sistema de control. A continuación se realiza una breve descripción de cada una de ellas.

- Capacidades de control del sistema:

- Sincronización de dispositivos: El sistema debe conocer el número de dispositivos disponibles y sincronizarse con ellos antes de proceder a la resolución del problema. Esto exige la existencia de un sistema de control centralizado que gestione ciertos aspectos del sistema.
- Mapeo del entorno: El sistema debe ser capaz de almacenar información del entorno, con el fin de monitorizar en todo momento de la forma más precisa posible el estado de los distintos elementos que forman parte del sistema y del entorno en el que se encuentra.
- Detección de elementos: El sistema debe ser capaz de obtener nuevo conocimiento del entorno a través de los distintos dispositivos (robots), este conocimiento puede realizar modificaciones sobre la información conocida previamente acerca del entorno.

- Capacidades de control de dispositivos:
  - Control centralizado: El sistema de control central, debe poder enviar mensajes a los robots para que ejecuten acciones de bajo nivel, las cuales podrán dividirse en primitivas de control más sencillas a la hora de ser aplicadas por cada uno de los diferentes robots.
  - Gestión de sensores: Cada robot debe ser capaz de obtener información a través de los distintos sensores que posee y gestionarla de manera independiente al resto del sistema, incluso enviar información de este tipo al sistema de control central.
  - Gestión de actuadores: Cada robot debe poder controlar los distintos actuadores que tenga disponibles de forma precisa, teniendo en cuenta sus capacidades.

- Capacidades de comunicación:
  - Gestión de mensajes: Cualquier dispositivo que forme parte del sistema debe ser capaz de comunicarse con el sistema de control central, es decir enviar y recibir mensajes.

- Capacidades de planificación

  - Planificación de tareas: El sistema debe ser capaz de resolver un problema de planificación automática, el cual se corresponde de manera parcial por la situación del entorno. Es decir existe parte de información que no es conocida por el sistema y que no se representa inicialmente en los ficheros que utiliza el planificador para generar la solución inicial.
  - Análisis del lenguaje: El sistema debe ser capaz de tratar la información en el lenguaje utilizado por el sistema de planificación automática, en este caso PDDL.

- Capacidades de configuración:

  - Configuración de parámetros: Existen ciertos elementos del sistema que deben poder ser configurados con el fin de aumentar el grado de variabilidad de los experimentos.
  - Portabilidad: El sistema debe poder ser ejecutado en otros ordenadores, siempre y cuando se cumplan las restricciones básicas.

Este conjunto de funcionalidades fueron definidas al comienzo del desarrollo del proyecto, el objetivo principal era conseguir que el sistema de control pudiera llevar a cabo todas estas funcionalidades, pero teniendo en cuenta algunas de las restricciones que se describen a continuación.

## B.2. Restricciones de funcionamiento

En este apartado se realiza una descripción de las distintas restricciones generales del sistema (que podrían ser consideradas como los requisitos de restricción de un análisis realizado de forma más extensa). En el caso de este proyecto las restricciones han sido divididas en dos grupos, aquellas que son debidas al hardware utilizado en este proyecto y las que son debidas a las librerías o *frameworks* de trabajo utilizados para el desarrollo del sistema de control.

## Restricciones Hardware

Las restricciones hardware se corresponden con las impuestas por los distintos dispositivos, que van a ser utilizados en este proyecto y consisten en presentar las distintas capacidades que se encuentran disponible en los distintos robots que van a ser utilizados.

- La conexión con el robot P3-DX, sólo podrá ser realizada mediante la utilización de un cable de conexión COM/USB, por lo que deberá de incluirse una CPU de control o un ordenador portátil sobre el robot que permita la ejecución del controlador.

- La conexión con el robot Lego NXT, se realizará mediante la tecnología *Bluetooth*, esto supone que la CPU en el cual se ejecute el sistema de control deberá de disponer de un sistema *Bluetooth* para poder enviar y recibir información del robot.

- El robot P3-DX, dispone de un anillo sonar de 8 transductores, dos sistemas de sensores de presión formados por 5 unidades cada uno (dispuestos en la parte delantera y trasera del robot) y dos motores, uno por cada una de las ruedas motrices.

- El robot Lego NXT, esta formado por tres *servos* de baja precisión que controlan los movimientos del brazo y un sensor de presión para detectar la recogida de los objetos.

## Restricciones Software

La restricciones software ser corresponden con aquellas limitaciones que deben ser cumplidas por el sistema de control. Estas en parte han surgido debido al sistema operativo o al *framework* de trabajo que ha sido seleccionado para el desarrollo del sistema de control.

- El lenguaje de programación en el cual será desarrollado el sistema de control será C#, este lenguaje ha sido parcialmente impuesto por el *framework* seleccionado para el desarrollo.

- Los ficheros de descripción del problema y el dominio que serán utilizados por el planificador, deben estar codificados en el lenguaje PDDL.

- El sistema de control debe ser ejecutado sobre un sistema operativo Windows. La mejor recomendación seria la utilización del sistema operativo Windows XP SP3.

- El sistema operativo debe tener instalado el framework de desarrollo Microsoft Robotic Developer Studio 2008 R2, el cual ha sido utilizado para el desarrollo del sistema de control y es necesario para poder ejecutarlo.

- Los distintos elementos del sistema de control deben ser diseñados teniendo en cuenta que deben ser distribuidos en nodos DSS (Decentralized Software Services), debido a que esta es la unidad básica de programación de MRDS.

## B.3. Entorno Operacional

En este apartado se realiza una descripción del entorno operacional. En este caso se realiza una descripción de dos entornos, debido a que el sistema de control desarrollado tiene que ser ejecutado en un ordenador en el cual debe estar instalado un software especifico y en un entorno físico en el cual deben existir ciertos robots cuyos comportamientos sean dominados.

- Entorno operacional software

  El entorno operacional software para este proyecto, se refiere al sistema operativo y las distintas librerías que deben haber sido instaladas para poder ser ejecutado. Cabe destacar que teniendo en cuenta la tecnología que ha sido seleccionado para la construcción de la arquitectura de control, este podría ser ejecutado en un grupo de ordenadores. Por lo tanto cada uno

de los equipos en los cuales se vayan a ejecutar los servicios, debería tener instalado un sistema operativo windows XP SP3 y las siguientes librerías o programas.

- Microsoft Robotic Developer Studio 2008 R2 o superior.
- Microsoft .NET Framework 2.0 o superior.
- Gold Parser Library
- MindSqual Library 1.0 o superior

Es necesario destacar que teóricamente es posible ejecutar el sistema de control en los otros operativos de Microsoft, pero en alguno de ellos como Windows Vista, se producen errores de ejecución que no han sido observados en Windows XP. Por lo que mi recomendación es ejecutar el sistema de control en el entorno recomendado.

- Entorno operacional hardware

  El entorno operacional hardware se refiere a los distintos elementos hardware que son necesarios para poder desplegar la arquitectura de control, con los dos tipos de robots que han sido utilizados en este proyecto.

  - Robot Pioneer P3-DX, constituido por un motor, un anillo sonar y dos conjuntos de *bumpers*.
  - Un brazo robótico, construido mediante Lego NXT 2.0.
  - Al menos una CPU, en la cual se deben ejecutar el sistema de control central y los controladores reactivos de los dos robots utilizados. Lo ideal para este proyecto, sería que cada sistema utilizara una centralita independiente, lo cual permitiría explotar de forma más eficiente la arquitectura de control que ha sido desarrollado para este proyecto.

## B.4. Diagrama de Casos de Uso

En este apartado se presenta el diagrama de casos de usos que ha sido desarrollado en la fase de análisis de este proyecto, en el se describen las principales funcionalidades del sistema de control. Debido a que el sistema desarrollado es de tipo automático, los actores que aparecen en el diagrama se corresponden con los distintos dispositivos de alto nivel en los que se ejecutará el sistema de control. A continuación se realiza una breve descripción de cada uno de los actores que serán incluidos en el diagrama.

- Controlador Central: Se corresponde con el sistema de control central.
- Controlador Rover: Se corresponde con el controlador de tipo reactivo creado para el robot P3-DX.
- Controlador Brazo: Se corresponde con el controlador de tipo reactivo desarrollado para el robot Lego NXT.

### B.4.1. Descripción de elementos

Para la realización de la descripción textual de los distintos casos de uso, se han seleccionado una serie de elementos que describen cada uno de los casos de uso. A continuación se realiza una descripción del significado de cada uno de los elementos utilizados para la descripción de los casos de uso.

- Código: Identificación unívoca abreviada del caso de uso, se construye mediante CU seguido de un - y de tres dígitos. Por ejemplo CU-001.
- Nombre: Identificación extendida del caso de uso.
- Actores: Conjunto de entidades que interactúan con el caso de uso. El caso de uso representa un funcionalidad demandada por un actor.


Figura B.1: Diagrama de Casos de Uso

- Descripción: Se realiza una descripción básica de la funcionalidad o funcionalidades del caso de uso.

- Precondiciones y poscondiciones: Se definen las condiciones que deben cumplirse para poder realizar una operación, y el estado en el que queda el sistema tras realizar una operación.

### B.4.2. Descripción textual de Casos de Uso

En este apartado se realiza una descripción de cada uno de los casos de uso presentados en el diagrama de la figura B.1.

| **Caso de Uso** | |
|---|---|
| Código | CU-001 |
| Nombre | Acceder Datos Sonar |
| Actores | Controlador Rover |
| Descripción | Permite al sistema de control del rover, acceder a la información detectada por el sónar. |
| Precondiciones | El controlador del rover, debe haber creado una asociación con el servicio de gestión del sónar. |
| Poscondiciones | Se establece una conexión con el sónar del robot. A partir de este momento se recibirá un evento cada vez que el sónar realice una lectura. |

Tabla B.1: Caso de Uso (Acceder Datos Sonar)

| **Caso de Uso** | |
|---|---|
| Código | CU-002 |
| Nombre | Acceder Datos Bumper |
| Actores | Controlador Rover |
| Descripción | Permite al sistema de control del rover, acceder a la información detectada por los distintos sensores de presión. |
| Precondiciones | El controlador del rover, debe haber creado una asociación con el servicio de gestión de los *bumpers*. |
| Poscondiciones | Se establece una conexión con los *bumpers* del robot. A partir de este momento se recibirá un evento cada vez que un *bumper* sea presionado. |

Tabla B.2: Caso de Uso (Acceder Datos Bumper)

| **Caso de Uso** | |
|---|---|
| Código | CU-003 |
| Nombre | Acceder Motor |
| Actores | Controlador Rover |
| Descripción | Permite al sistema de control del rover, acceder al sistema de control de los motores. |
| Precondiciones | El controlador del rover, debe haber creado una asociación con el servicio de gestión del motor. |
| Poscondiciones | Se establece una conexión con los motores del robot. A partir de este momento el sistema podrá utilizar las funciones de control de los motores. |

Tabla B.3: Caso de Uso (Acceder Motor)

| **Caso de Uso** | |
|---|---|
| Código | CU-004 |
| Nombre | Mover Motor |
| Actores | Controlador Rover |
| Descripción | Envía un mensaje al sistema de control de motores, indicando la dirección y la velocidad a la que debe moverse. |
| Precondiciones | Se debe haber accedido al sistema de control del motor. |
| Poscondiciones | El robot ejecuta la acción solicitada, se moverá a velocidad constante en la dirección indicada. |

Tabla B.4: Caso de Uso (Mover Motor)

| **Caso de Uso** | |
|---|---|
| Código | CU-005 |
| Nombre | Girar Motor |
| Actores | Controlador Rover |
| Descripción | Envía un mensaje al sistema de control de motores, indicando los grados que debe girar sobre si mismo. |
| Precondiciones | Se debe haber accedido al sistema de control del motor. |
| Poscondiciones | El robot ejecuta la acción solicitada, girar sobre si mismo el número de grados solicitados. |

Tabla B.5: Caso de Uso (Girar Motor)

| **Caso de Uso** | |
|---|---|
| Código | CU-006 |
| Nombre | Detener Motor |
| Actores | Controlador Rover |
| Descripción | Envía un mensaje al sistema de control de motores, indicando que debe detener la acción que este realizando. |
| Precondiciones | Se debe haber accedido al sistema de control del motor. |
| Poscondiciones | El robot ejecuta la acción solicitada, dejará de ejecutar cualquier operación. |

Tabla B.6: Caso de Uso (Detener Motor)

| **Caso de Uso** | |
|---|---|
| Código | CU-007 |
| Nombre | Detener Servo |
| Actores | Controlador Brazo |
| Descripción | Envía un mensaje al sistema de control de motores, indicando que debe detener la acción que este realizando. |
| Precondiciones | Se debe haber accedido al sistema de control del motor. |
| Poscondiciones | El robot ejecuta la acción solicitada, se detendrá en la posición en la que se encuentre. |

Tabla B.7: Caso de Uso (Detener Motor)

| **Caso de Uso** | |
|---|---|
| Código | CU-008 |
| Nombre | Mover Servo |
| Actores | Controlador Brazo |
| Descripción | Envía un mensaje al sistema de control de motores, indicando la velocidad y la dirección. |
| Precondiciones | Se debe haber accedido al sistema de control del motor. |
| Poscondiciones | El robot ejecuta la acción solicitada, se moverá a velocidad constante en la dirección indicada. |

Tabla B.8: Caso de Uso (Mover Servo)

| **Caso de Uso** | |
|---|---|
| Código | CU-009 |
| Nombre | Girar Servo |
| Actores | Controlador Brazo |
| Descripción | Envía un mensaje al sistema de control de motores, indicando la velocidad de giro y la dirección. |
| Precondiciones | Se debe haber accedido al sistema de control del motor. |
| Poscondiciones | El robot ejecuta la acción solicitada, girar sobre si mismo a velocidad constante y en la dirección indicada. |

Tabla B.9: Caso de Uso (Girar Servo)

| **Caso de Uso** | |
|---|---|
| Código | CU-010 |
| Nombre | Enviar Respuesta |
| Actores | Controlador Central, Controlador Rover y Controlador Brazo |
| Descripción | Envía un mensaje de tipo respuesta, desde uno de los controladores a otro. |
| Precondiciones | Se debe haber establecido previamente una asociación entre los puntos de la comunicación. |
| Poscondiciones | Un mensaje de tipo respuesta es recibido por un controlador. |

Tabla B.10: Caso de Uso (Enviar Respuesta)

| **Caso de Uso** | |
|---|---|
| Código | CU-011 |
| Nombre | Enviar Petición |
| Actores | Controlador Central, Controlador Rover y Controlador Brazo |
| Descripción | Envía un mensaje de tipo petición, desde uno de los servicio del sistema a otro. |
| Precondiciones | Se debe haber establecidopreviamente una asociación entre los puntos de la comunicación. |
| Poscondiciones | Un mensaje de tipo petición es recibido por un servicio. |

Tabla B.11: Caso de Uso (Enviar Petición)

| Caso de Uso | |
|---|---|
| Código | CU-012 |
| Nombre | Gestionar Mensaje |
| Actores | Controlador Central, Controlador Rover y Controlador Brazo |
| Descripción | El sistema debe ser capaz de analizar los mensajes que envían los distintos controladores, obteniendo la información útil de ellos. |
| Precondiciones | Se debe haber recibido un mensaje de otro servicio. |
| Poscondiciones | Se obtiene la información contenida en el mensaje. |

Tabla B.12: Caso de Uso (Gestionar Mensaje)

| Caso de Uso | |
|---|---|
| Código | CU-013 |
| Nombre | Crear Mapa |
| Actores | Controlador Central |
| Descripción | El sistema creará un mapa de alto nivel, en el cual representará el entorno en el que se encuentran los distintos robots. |
| Precondiciones | Debe existir un medio para la obtención de la información mediante la cual se creará el mapa. |
| Poscondiciones | El sistema generará un mapa de alto nivel. |

Tabla B.13: Caso de Uso (Crear Mapa)

| Caso de Uso | |
|---|---|
| Código | CU-014 |
| Nombre | Actualizar Mapa |
| Actores | Controlador Central |
| Descripción | El sistema ofrecerá una serie de funcionalidades que permitan modificar la información del mapa de alto nivel. |
| Precondiciones | Debe existe un mapa de alto nivel previo. |
| Poscondiciones | El mapa de alto nivel es modificado. |

Tabla B.14: Caso de Uso (Actualizar Mapa)

| Caso de Uso | |
|---|---|
| Código | CU-015 |
| Nombre | Transformar Mapa |
| Actores | Controlador Central |
| Descripción | El sistema debe permitir realizar una transformación del mapa de alto nivel, al lenguaje de representación del planificador. |
| Precondiciones | Debe existe un mapa de alto nivel previo. |
| Poscondiciones | Se obtiene el estado del entorno en el lenguaje de representación del planificador. |

Tabla B.15: Caso de Uso (Transformar Mapa)

| **Caso de Uso** | |
|---|---|
| Código | CU-016 |
| Nombre | Cargar Configuración |
| Actores | Controlador Central |
| Descripción | El sistema debe permitir cargar una serie de información de configuración a través de un fichero. |
| Precondiciones | Debe existir un fichero previo con la información de la configuración. |
| Poscondiciones | Se carga la configuración en el sistema. |

Tabla B.16: Caso de Uso (Cargar Configuración)

| **Caso de Uso** | |
|---|---|
| Código | CU-017 |
| Nombre | Planificar |
| Actores | Controlador Central |
| Descripción | El sistema debe ser capaz de utilizar un planificador y obtener un solución a un determinado problema. |
| Precondiciones | Debe existir un planificador que puede utilizar el sistema, y deben existir un problema a resolver y una representación del dominio. |
| Poscondiciones | Se obtiene un conjunto secuencial de acciones que describen las tareas a realizar por los distintos robots. |

Tabla B.17: Caso de Uso (Planificador)

| Caso de Uso | |
|---|---|
| Código | CU-018 |
| Nombre | Cargar Problema |
| Actores | Controlador Central |
| Descripción | El sistema debe poder cargar el problema a resolver en lenguaje PDDL. |
| Precondiciones | Debe existir el fichero que contiene el problema. |
| Poscondiciones | Se carga la información del problema. |

Tabla B.18: Caso de Uso (Cargar Problema)

| Caso de Uso | |
|---|---|
| Código | CU-019 |
| Nombre | Gestionar Representación Simbólica |
| Actores | Controlador Central |
| Descripción | El sistema debe ser capaz de almacenar y gestionar la representación simbólica del entorno, con el fin de poder interactuar de manera más precisa con el planificador. |
| Precondiciones | Debe existir un representación del dominio y del problema a resolver. |
| Poscondiciones | Se obtiene un fichero de la representación simbólica que puede utilizar el planificador. |

Tabla B.19: Caso de Uso (Gestionar Representación Simbólica)

| Caso de Uso | |
|---|---|
| Código | CU-020 |
| Nombre | Gestionar Representación Simbólica |
| Actores | Cargar Dominio |
| Descripción | El sistema debe poder cargar el dominio a resolver en lenguaje PDDL desde un fichero. |
| Precondiciones | Debe existir el fichero que contiene el dominio. |
| Poscondiciones | Se carga la información del problema. |

Tabla B.20: Caso de Uso (Cargar Dominio)

| Caso de Uso | |
|---|---|
| Código | CU-021 |
| Nombre | Analizador fichero PDDL |
| Actores | Cargar Dominio |
| Descripción | El sistema debe ser capaz de analizar léxica, sintáctica y semántica los ficheros PDDL. |
| Precondiciones | Debe existir el fichero en PDDL. |
| Poscondiciones | Se genera una serie de estructuras con la información almacenada en el fichero. |

Tabla B.21: Caso de Uso (Analizador fichero PDDL)

# B.5. Diseño del sistema

## B.5.1. Diagrama de componentes

El diseño arquitectónico del sistema se describe mediante la presentación de un diagrama de componentes formado por tres módulos, que se corresponde con los tres sistema de control que han sido descritos en el tercer capítulo de este documento. Como se puede observar en la figura B.2, se presenta un diagrama de despliegue que describe como se distribuyen los distintos componentes entre los módulos y como interactúan entre sí, mediante los distintos sistemas de comunicación utilizados por el sistema de control.

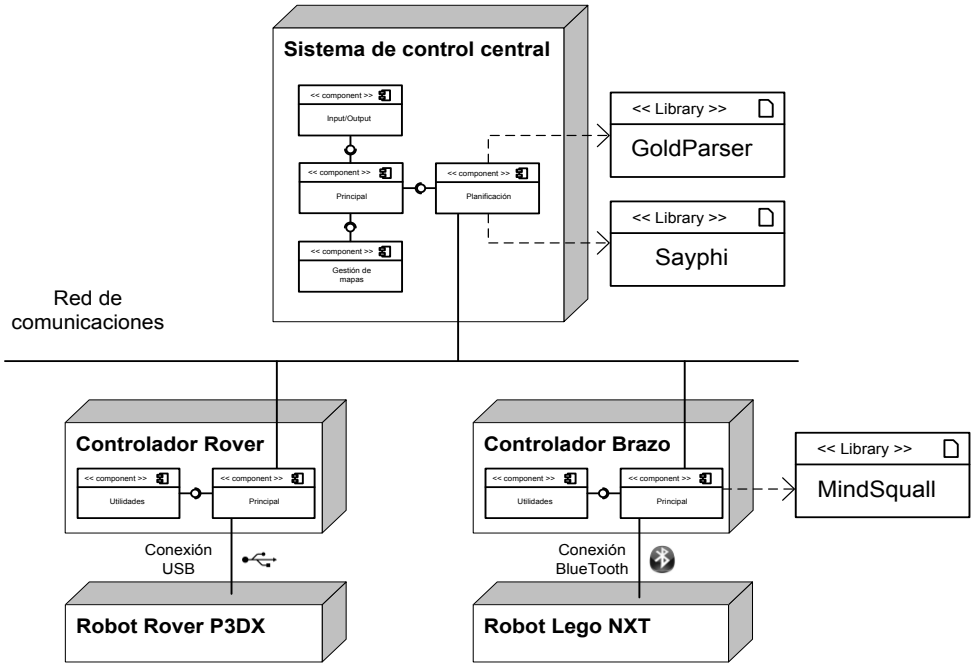


Figura B.2: Diagrama de Componentes del sistema

El primer nodo del sistema se corresponde con el sistema de control central, este se encuentra formado por 4 componentes.

- Principal: Este componente se corresponde como el proceso principal del sis-

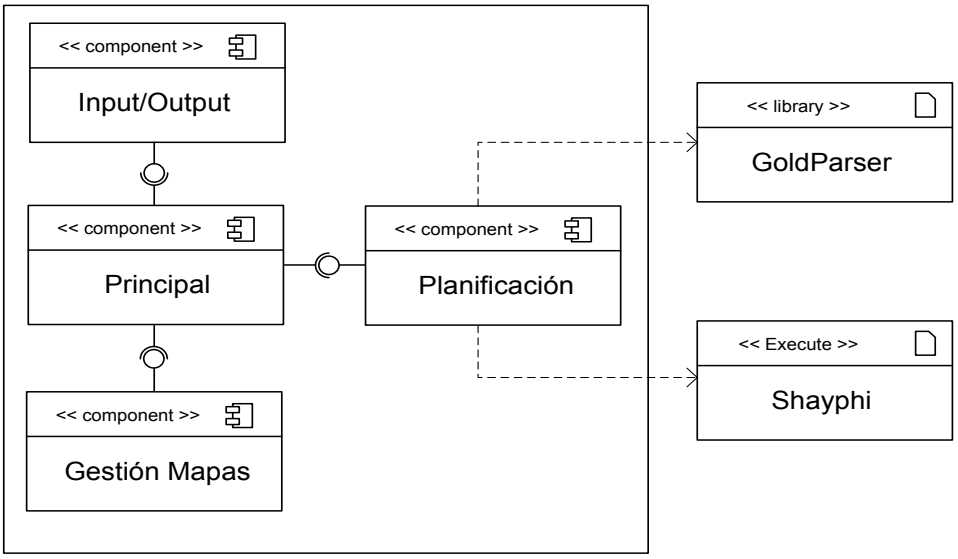


Figura B.3: Diagrama de Componentes del sistema de control central

tema de control central, este se encarga de la comunicación con los sistemas de control reactivo de los robots, así como la gestión del sistema de planificación, la entrada/salida y la gestión y generación de los mapas abstractos de conocimiento.

- Planificación: Este componente se encarga de gestionar los procesos de interacción con el planificador, se ocupa de la gestión de los sistemas de análisis de los ficheros de definición del dominio y el problema, así como del almacenamiento y manipulación de la información en representación simbólica.

- Gestor de Mapas: Este componente se encarga de la gestión del sistema de representación del entorno de bajo nivel. Además, gestiona el proceso de monitorización ya que almacena los cambios en el entorno en tiempo real.

- InputOutput: Este componente se encarga de la gestión de los datos de entrada y salida del sistema de control al planificador Sayphi [26], así como la lectura de los ficheros de configuración del sistema.

- GoldParser: Este artefacto es una librería dinámica utilizada para el desar-

rollo de los sistemas de análisis de los ficheros de definición del dominio y el problema.

- Sayphi: Este artefacto se corresponde con el planificador que ha sido elegido y con el cual interactúa el sistema para realizar los procesos de planificación automática.

El resto de los nodos presentado en la figura B.2, son similares entre sí y representan los controladores reactivos de los robots, donde la única diferencia existente entre ellos es la utilización de la librería MindSquall en el controlador del robot Lego NXT y en algunas funciones de las clases del componente principal, pero a nivel arquitectónico son completamente similares. Por lo que sólo se presenta el diagrama de componentes para el robot Lego NXT, en la figura B.3.

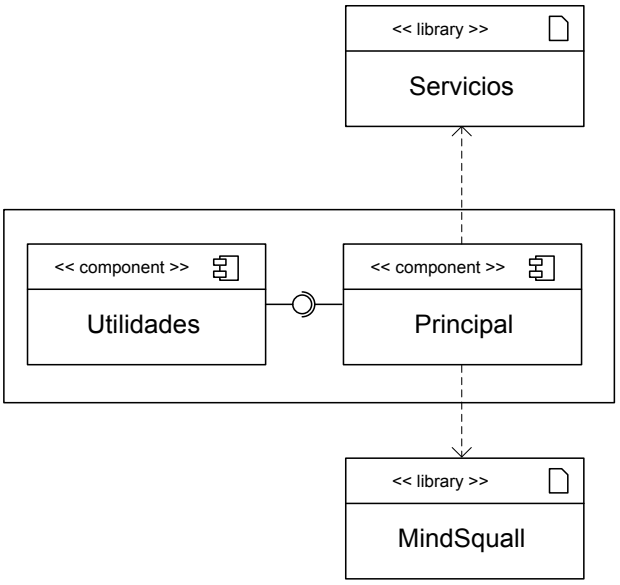


Figura B.4: Diagrama de Componentes del sistema de control del brazo NXT

- Principal: Este componente se corresponde con el proceso principal del sistema de control de cada uno de los robots. Este se encarga de la comunicación con el sistema de control central y la interacción con los distintos sensores y actuadores disponibles en cada uno de los robots.

- Utilidades: Este componente se encarga de gestionar las funciones genéricas utilizadas por los sistemas de control de los robots, como por ejemplo la gestión de los límites de lectura del sonar, o los procesos de transformación de los datos enviados o recibidos desde el sistema de control central.

- MindSquall: Este artefacto es una librería dinámica utilizada para interactuar con el robot Lego NXT, esta librería fue incluida en la fase final de diseño debido a los distintos problemas que surgieron con los servicios que ofrecía MRDS.

- Servicios: Este artefacto es de tipo virtual, debido a que no corresponde con una librería o aplicación, sino que representación la conexión que se produce entre el controlador y los sensores y actuadores de los distintos robots.

## B.5.2. Diagrama de clases

En este apartado se realiza la explosión de los distintos componentes que forman los nodos del sistema, para cada uno de los componentes se presenta el diagrama de clases, en el cual se definen las clases que serán desarrolladas en el proyecto y las relaciones que existen entre ellas. Cabe destacar de los diagramas de clases presentados en este apartado no son detallados, de forma que sólo se presentan las clases, las relaciones que existen entre ellas y los atributos más destacados de algunas de las clases.

### B.5.2.1. Diagramas de clases Sistema de control central

A continuación se presenta los distintos diagramas de clases definidos para el nodo correspondiente al controlador central, debido a que clases de todos los componentes interactúan con una clase que se encuentra en el componente principal, esta será incluida en todos los diagramas con el fin de describir las relaciones de esta clases con todas las demás. A continuación en la figura B.4, se presenta el diagrama de clases del componente más importante del sistema, denominado

**principal**, ya que este se encarga de la gestión de los principales elementos del sistema de control central de la arquitectura de control descrita en este proyecto.

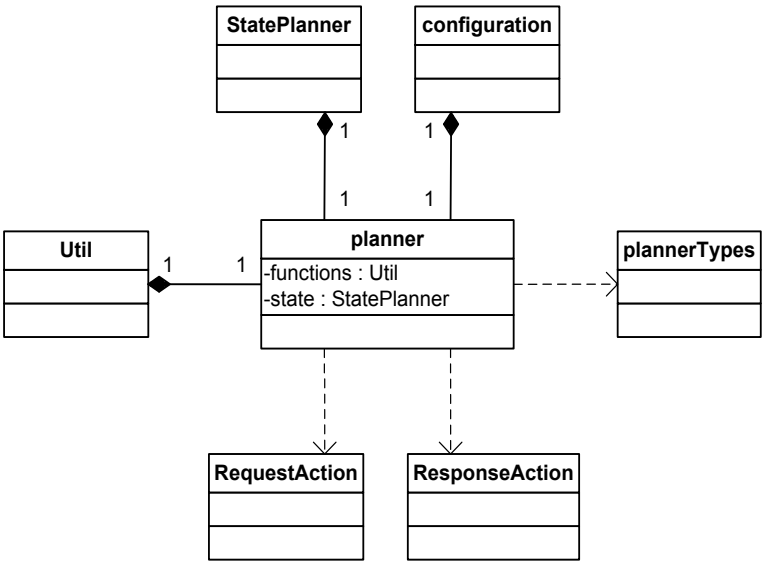


Figura B.5: Diagrama de clases del componente principal

- planner: Es la clase principal del sistema, su función consiste en coordinar todo el sistema de control. Gestiona el proceso de planificación, la monitorización del entorno a través del mapa abstracto, la sincronización de las tareas y los procesos de comunicación.

- plannerTypes: Es una clase de tipo definición, obligatoria para todo DSS, sirve para definir las funcionalidades que ofrece un servicios a otros cuando se produce una asociación entre servicios.

- statePlanner: Esta clase define el estado del planificador, en ella se almacena información referente al estado del autómata de control y otros parámetros especiales del sistema de control central.

- util: En esta clase se encuentran desarrolladas un conjunto de funciones para la gestión de los datos de comunicación, así como otras funciones auxiliares que se utilizan en el sistema.

- RequestAction: Mediante esta clase se define la estructura y los métodos de los mensajes de tipo petición, que se intercambian entre los robots y el sistema de control central.

- ResponseAction: Esta clase define la estructura y los métodos de los mensajes de tipo respuesta, que son intercambiados entre el sistema de control central y los distintos robots.

- configuration: Esta clase ha sido implementada mediante el patrón de diseño *Singleton* y gestiona la información almacenada en el fichero de configuración cargado al comienzo del proceso de ejecución.

En la figura B.6, se presenta el diagrama de clases correspondiente al componente de **gestión de mapas**, en el se gestiona toda la funcionalidad referente a la gestión del mapa abstracto y la monitorización del entorno.

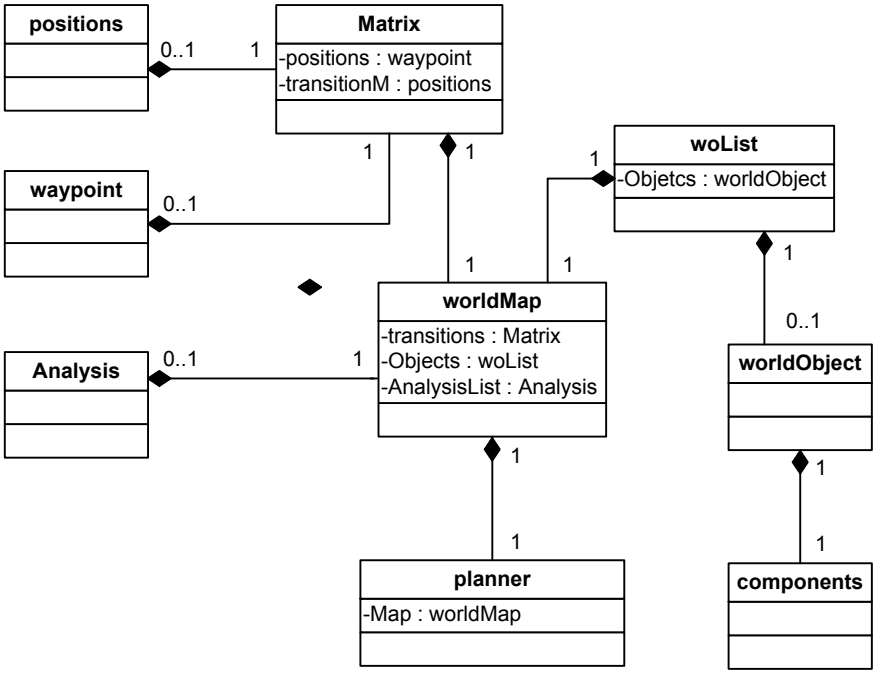


Figura B.6: Diagrama de clases del componente gestión de mapas

- matrix: Esta clase define la matriz de transiciones entre los distintos puntos del mapa (waypoints), esta matriz, es construida por nodos de tipo position,

también incluye una lista de *waypoint*, con el fin de establecer una relación entre el nombre del *waypoint* según la representación simbólica y la posición en la matriz.

- positions: Mediante esta clase se define las propiedades de cada uno de los elementos de la matriz de transiciones implementada en la clase Matrix, almacena la dirección y la visibilidad que existe de un *waypoint* del mapa a otro.

- waypoint: Mediante esta clase se almacenan los distintos *waypoint* que existen en el mapa, indicándose si existen objetos en ellos, así como su posición en la Matriz de transiciones.

- Analysis: Mediante esta clase se definen los distintos análisis que son efectuados por los robots de tipo transportador, ha sido definida de forma independiente ya que los análisis pueden ser enviados a través del sistema de comunicaciones.

- woList: Mediante esta clase se define todos los elementos que gestionan los objetos que se encuentran en el dominio. Sólo han sido considerados como objetos los diferentes robots que se encuentran en el entorno.

- worldObject: Esta clase ha sido definida para almacenar la información referente a los distintos dispositivos (robots) que se encuentran en el dominio.

- componentes: Mediante esta clase se realiza la definición de las cualidades específicas de los distintos robots, inicialmente se ha utilizado una única clase debido a las similitudes entre los distintos robots utilizados, pero de esta manera en el caso de tener que incluir nuevas clases de componentes, se podría crear una clase abstracta sin realizar muchas modificaciones en el código fuente.

- worldMap: Mediante esta clase se gestionan todos los procesos del mapa abstracto del entorno, se controla el proceso de monitorización y actualización

de la representación simbólica.

En la figura B.7, se presenta el diagrama de clases correspondiente al componente de **input/output**, mediante este componente se realiza la gestión de los fichero de entrada y salida utilizados por el sistema.

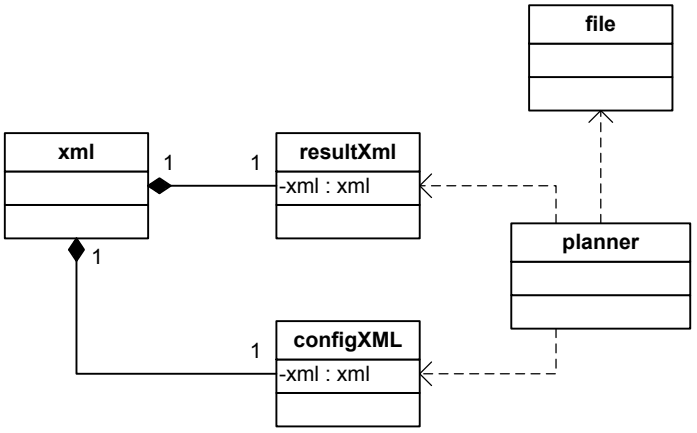


Figura B.7: Diagrama de clases del componente Input Output

- xml: Es una clase genérica que define los métodos básicos para la lectura y escritura de ficheros de tipo XML, en esta clase se definen funciones de búsqueda de tags y gestión de listas de nodos xml.

- resultXml: Esta clase es utilizada para cargar el fichero de salida del planificador Sayphi, permite la gestión de forma sencilla del fichero xml en el cual se presentan el conjunto secuencial de acciones o tareas necesarios para resolver el problema.

- configXml: Esta clase gestiona la lectura de los ficheros de configuración que utiliza el sistema de control.

- file: Esta clase permite la lectura y escritura de ficheros de texto, es utilizada para la generación de información referente al proceso de ejecución.

En la figura B.8, se presenta el diagrama de clases correspondiente al componente de **planificación**, en el se presenta las clases que gestionan parcialmente el proceso de planificación y que almacenan la representación simbólico del entorno.

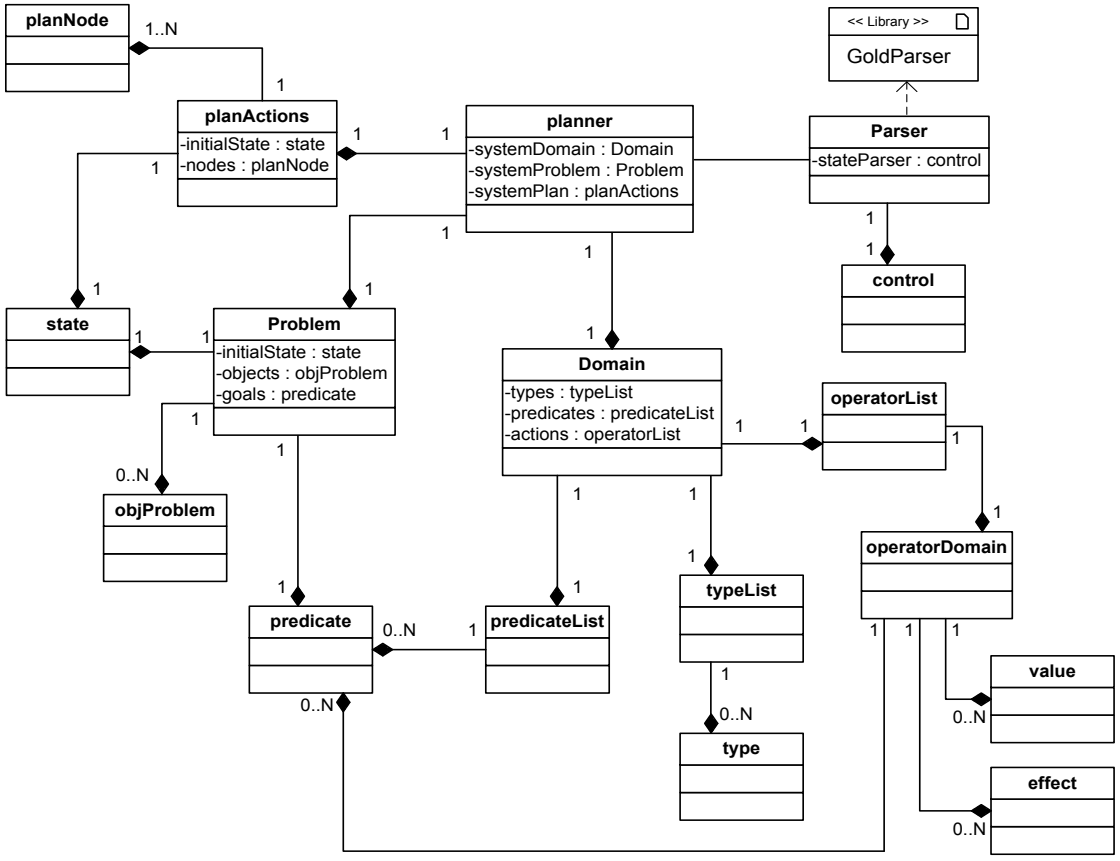


Figura B.8: Diagrama de clases del componente planificación

- domain: Mediante esta clase se define el dominio de planificación, en ella se almacena todos los elementos que forman un dominio de planificación, los tipos de objetos, la estructura de los predicados que define las características de los estados y los operadores que pueden ser aplicados sobre un estado para producir una transición.

- predicateList: En esta clase se definen los procesos de gestión de un conjunto de predicados, son utilizados para representar los predicados que forman parte del dominio o que forman parte de un estado.

- predicate: En esta clase se define la estructura de un predicado, para este se almacenan el nombre del predicado y el valor de cada una de las variables del predicado.

- typeList: Esta clase implementa los procesos de gestión de la lista de tipos de objetos del dominio.

- type: Esta clase engloba la estructura y la operaciones básicas de un tipo de objeto del dominio.

- operatorList: Mediante esta clase se gestionan los procesos de creación y aplicación de los operadores definidos en el fichero del dominio.

- operatorDomain: Esta clase, se utiliza para definir un operador en representación simbólica, este está formando por una lista de efectos y una lista de variables.

- effect: Mediante esta clase se define la lista de efectos que se producen tras la aplicación de un operador.

- value: Esta clase define la estructura de los parámetros que son definidos en los operadores del dominio.

- parser: En esta clase se define el proceso de análisis de los ficheros en PDDL, en ella se encuentra implementado un autómata finito mediante el cual se realiza el análisis de los ficheros que definen el dominio y el problema a resolver.

- control: Esta clase ha sido definida para almacenar el estado del proceso de análisis de los ficheros PDDL, además incluye funciones de gestión que son utilizadas dicho proceso.

- problem: En este clase se define el problema a resolver. En el se incluyen todos los procesos de gestión, creación y eliminación de los elementos del

problema, esta clase es utilizada para la definición del fichero del problema a resolver, en los proceso de planificación y replanificación.

- state: Esta clase, es utilizada para definir un estado del entorno en representación simbólica.
- planActions: Esta clase es utilizada para almacenar y gestionar la lista de acciones que deben ser realizadas para completar el plan. Mediante ella también se realiza parte de la monitorización teniendo en cuenta las acciones que han sido ejecutadas correctamente.
- planNode: Esta clase define una acción del plan secuencial de acciones que ha sido generado por el proceso de planificación.

#### B.5.2.2. Diagramas de clases Sistema de control del robot P3-DX

En la figura B.9, se presenta el diagrama de clases del sistema de control para los robot pioneer P3-DX, debido a la simplicidad del modelo de clases, este va a ser presentado en un sólo diagrama a pesar de que su representación arquitectónica está formada por dos componentes.

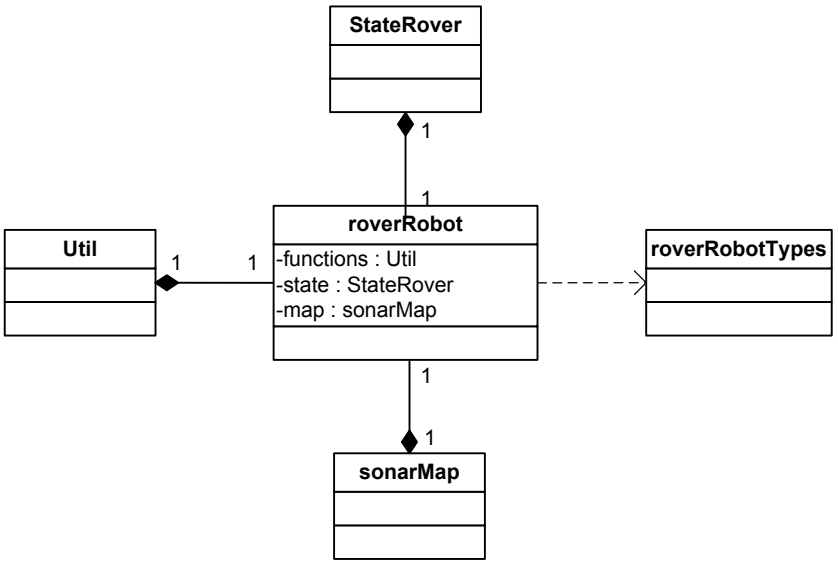


Figura B.9: Diagrama de clases del robot rover P3-DX

- roverRobot: Esta es la clase principal del sistema de control del robot P3-DX, en ella se define las asociaciones con los distintos servicios, los manejadores que gestionan los eventos producidos por los socios, el proceso de gestión del autómata finito de control y los métodos referentes a los procesos de comunicación.

- stateRobot: Esta clase almacena toda la información referente al sistema de control del robot, como el estado del autómata finito que gestiona el proceso de ejecución, así como los distintos métodos para manipular los valores almacenados.

- roverRobotTypes: Es una clase de tipo definición, obligatoria para todo DSS, sirve para definir las funcionalidades que ofrece un servicios a otros cuando se produce una asociación entre servicios.

- util: Esta es una clase en la cual se encuentran desarrolladas un conjunto de funciones para la gestión de los datos de comunicación, así como otras funciones que se utilizan en el sistema.

- sonarMap: En esta clase se define un conjunto de métodos y estructuras para gestionar la información obtenida a través del sonar, esta clase ha sido definida para mejorar el procesado de la información obtenida a través de este tipo de sensor.

#### B.5.2.3. Diagramas de clases Sistema de control del robot NXT

En la figura B.10, se presenta el diagrama del clases del sistema de control para el robot Lego NXT, al igual que en el caso anterior, las distintas clases que han sido definidas para la gestión del robot se presentan en un único diagrama.

- armRobot: Esta es la clase principal del sistema de control del robot Lego NXT, en ella se define las asociaciones con los distintos servicios, los manejadores que gestionan los eventos producidos por los socios, el proceso de

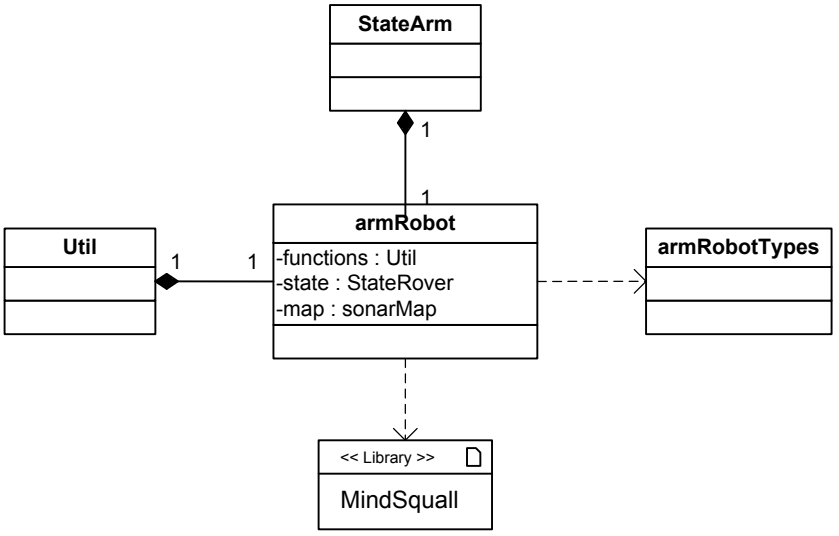


Figura B.10: Diagrama de clases del robot Lego NXT

gestión del autómata finito de control y los métodos referentes a los procesos de comunicación.

- stateArm: Esta clase almacena toda la información referente al sistema de control del robot, como el estado del autómata finito que gestiona el proceso de ejecución, así como los distintos métodos para manipular los valores almacenados.

- armRobotTypes: Es una clase de definición, obligatoria para todo DSS, sirve para definir las funcionalidades que ofrece un servicios a otros cuando se produce una asociación entre servicios.

- util: Esta es una clase en la cual se encuentran desarrolladas un conjunto de funciones para la gestión de los datos de comunicación, así como otras funciones que se utilizan en el sistema.

# Apéndice C

# Analizador del lenguaje PDDL

En este anexo se realiza una descripción detallada del analizador del lenguaje PDDL, que ha sido desarrollado para este proyecto. En este apartado se realiza una descripción detallada de los elementos del analizador que han sido utilizados en el sistema de control y se presentan las distintas gramáticas que han sido diseñadas.

Una gramática formal, según Chomsky [31], puede ser definida mediante una cuádrupla de la siguiente forma,

$$G = (N, T, S, P) \tag{C.1}$$

- N, representa a un conjunto de símbolos no terminales, los cuales pueden ser considerados como variables, cuyo valor depende de como sea recorrida la gramática al analizar los ejemplos.

- T, representa a un conjunto de símbolos terminales o constantes, el cual debe ser disjunto con el conjunto N.

- S, representa a un único símbolo del conjunto N, el cual define el símbolo inicial de la gramática.

- P, es un conjunto formado por las distintos reglas de producción, de la siguiente forma

$$(N \cap T)^* N (N \cap T)* \rightarrow (N \cap T)^* \quad \text{(C.2)}$$

donde * es la clausura de Kleene, de forma que cada regla de producción realiza un mapeo de una cadena de símbolos a otra, donde la primera cadena siempre contiene al menos un símbolo no terminal, que permite la navegación por las distintas producciones.

## C.1. Análisis léxico

El análisis léxico es la primera de la etapas que posee un procesador de lenguajes, el proceso de análisis léxico ha sido realizado mediante las herramientas que ofrece GoldParser [30], mediante la creación de un fichero de definición en el cual se incluyen los distintos tokens que definen el lenguaje o lenguajes que tienen que ser procesados. A continuación se presentan dos tablas con los distintos tokens que han sido utilizados para definir las dos gramáticas.

| **Listado de tokens del Dominio** | | | |
|---|---|---|---|
| ID | VALUE_ID | ( | ) |
| define | domain | : | types |
| predicates | not | action | parameters |
| precondition | effect | and | - |

Tabla C.1: Tabla de tokens del Dominio

| **Listado de tokens del Problema** | | | |
|---|---|---|---|
| ID | VALUE_ID | ( | ) |
| define | problem | domain | : |
| objects | and | init | goal |

Tabla C.2: Tabla de tokens del Problema

## C.2. Análisis sintáctico

El análisis sintáctico es la segunda de la etapas que posee un procesador de lenguajes o compilador, para construir el analizar sintáctico se ha realizado la definición de dos gramáticas en formato BNF (Backus-Naur Form o Backus Normal Form), cada una de ellas está formada por los distintos tokens o palabras reservadas que ha sido presentado en el apartado anterior de este anexo. A continuación se presentan las dos gramáticas.

La primera de ellas ha sido desarrollada para el análisis del fichero que almacena la información del dominio PDDL, donde el símbolo de inicio es PROGRAM.

```
<PROGRAM> ::= '(' 'define' <DOMAIN> <TYPESLIST> <PREDICATES> <ACTIONS> ')'

<DOMAIN> ::= '(' 'domain' ID ')'

<TYPESLIST> ::= '(' ':' 'types' <TYPES> ')'

<TYPES> ::= ID <TYPES>
| ID

<PREDICATES> ::= '(' ':' 'predicates' <PREDICATES_LIST> ')'

<PREDICATES_LIST> ::= <PREDICATE> <PREDICATES_LIST>
| <PREDICATE>

<PREDICATE> ::= '(' ID <VALUES_PREDICATE> ')'
| '(' 'not' '(' ID <VALUES_PREDICATE> ')' ')'

<VALUES_PREDICATE>  ::= <VALUE_PREDICATE> <VALUES_PREDICATE>
| <VALUE_PREDICATE>
```

```
<VALUE_PREDICATE>   ::= VALUE_ID '-' ID
| VALUE_ID

<ACTIONS> := <ACTION> <ACTIONS>
| <ACTION>

<ACTION> ::= '(' <ACTION_ID> <PARAMETERS> <CONDITIONS> <EFFECTS> ')'

<ACTION_ID> ::= ':' 'action' ID

<PARAMETERS> ::= ':' 'parameters' '(' <VALUES> ')'

<VALUES> ::= <VALUE> <VALUES>
| <VALUE>

<VALUE> ::= VALUE_ID '-' ID

<CONDITIONS> ::= ':' 'precondition' <CONDITIONS_LIST>

<CONDITIONS_LIST> ::= '(' 'and' <PREDICATES_LIST> ')'
| <PREDICATE>

<EFFECTS> ::= ':' 'effect' <EFFECTS_LIST>

<EFFECTS_LIST> ::= '(' 'and' <PREDICATES_LIST> ')'
| <PREDICATE>
```

La segunda gramática presentada en este anexo se corresponde con la utilizada para el análisis de los ficheros que almacenan la información de los problemas que deben ser resueltos por el planificador, al igual que en la anterior gramática,

el símbolo de inicio es PROGRAM.

```
<PROGRAM> ::= '(' 'define' <NAME> <DOMAIN> <OBJECTS> <INIT> <GOALS> ')'

<NAME> ::= '(' 'problem' VALUE_ID ')'

<DOMAIN> ::= '(' ':' 'domain' VALUE_ID ')'

<OBJECTS> ::= '(' ':' 'objects' <TYPES> ')'

<TYPES> ::= <TYPE> <TYPES>
| <TYPE>

<TYPE> ::= <ELEMENTS> CONECTOR VALUE_ID

<ELEMENTS> ::= VALUE_ID <ELEMENTS>
| VALUE_ID

<INIT> ::= '(' ':' 'init' <PREDICATES> ')'

<PREDICATES> ::= <PREDICATE> <PREDICATES>
| <PREDICATE>

<PREDICATE> ::= '(' VALUE_ID <VALUES> ')'

<VALUES> ::= VALUE_ID <VALUES>
| VALUE_ID

<GOALS> ::= '(' ':' 'goal' <SETGOALS> ')'

<SETGOALS> ::= '(' 'and' <PREDICATES> ')'
```

## C.3. Análisis semántico

El análisis semántico es la tercera de las etapas que posee el analizador, en este caso el proceso de análisis semántico se realiza de forma paralela al análisis sintáctico, ya que el autómata de control que ofrece GoldParser para la realización del análisis semántico se ejecuta de esta manera, pasando por los distintos estados presentados en el autómata de la figura C.1.

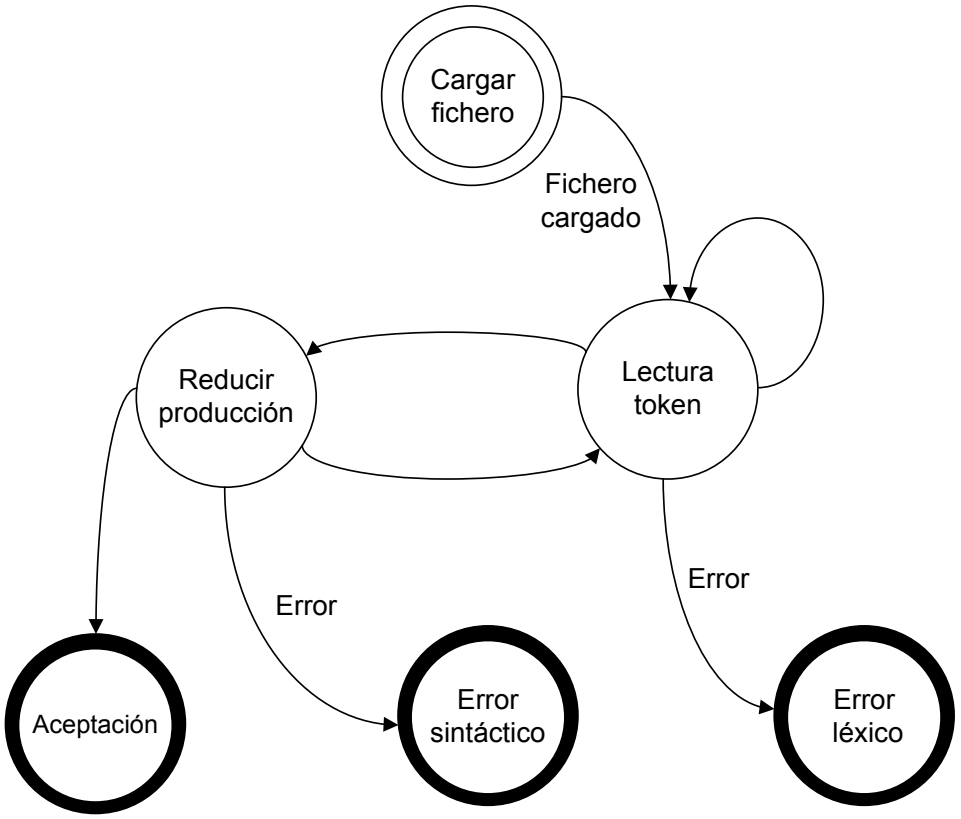


Figura C.1: Autómata del sistema de análisis

- Carga fichero: Es el estado inicial del proceso de análisis y consiste en cargar los fichero preprocesados que contiene los autómatas que serán utilizados durante el análisis léxico y sintáctico y el fichero que debe ser analizado.

- Error léxico: El proceso llega a este estado cuando se ha detectado un token que no pertenecer a la gramática, supone la finalización del proceso de análisis indicándose que se se ha producido un error.

- Error sintáctico: El proceso llega a este estado cuando se ha detectado un token que no pertenecer a la gramática, supone la finalización del proceso de análisis indicándose que se se ha producido un error.

- Lectura de tokens: Mediante este estado se realiza la lectura de un token del fichero de entrada, durante su ejecución se realiza el almacenamiento local del token leido y el control de su posición en la gramática, de forma que se pueda detectar el significado semántico de los tokens que están siendo recibidos.

- Reducción de reglas: Mediante este estado se produce la reducción de una de las reglas de la gramática, cuando se llega a este estado normalmente se ejecuta un proceso que genera un predicado, un conjunto de variables con tipo o alguno de los elementos de un operador en el modelo de representación simbólica.

- Aceptación: Este es el estado final del sistema, se llega a él cuando el fichero ha sido analizado correctamente y no se ha detectado ningún error.

El funcionamiento del proceso de análisis del lenguaje se basa en la reducción de las reglas que forman la gramática. El sistema almacena en una estructura de datos auxiliar todos los tokens que son recogidos por el analizador léxico y cuando una producción o regla es reducida por el proceso de análisis sintáctico, se comprueba en una tabla si la regla contiene información que es útil para el sistema. En caso afirmativo el sistema almacena la información en la estructura que corresponda teniendo en cuenta el significa semántico de la producción y elimina los tokens de la estructura auxiliar de almancenamiento. En caso contrario sólo se produce la eliminación de los tokens de la estructura auxiliar. Si se produzca un error durante el proceso de análisis de alguno de los ficheros, el sistema de

control finalizará su ejecución debido a que no ha podido procesar la información conocida del entorno.

# Apéndice D

# Código PDDL

En este anexo se presenta el código en PDDL del dominio y los problemas que han sido presentados en el capítulo 4 de este documento.

## D.1. Dominio

```
(define
(domain Rovers)
(:types rover arm waypoint direction position object lander chargeZone)
(:predicates
(at_rover ?r - rover ?y - waypoint)
(can_traverse ?r - rover ?x - waypoint ?y - waypoint ?d - direction)
(position_rover ?r - rover ?d - direction)
(available_rover ?r - rover)
(have_store ?r - rover)
(have_data_store ?r - rover)
(empty_store ?r - rover)
(visible ?w - waypoint ?p - waypoint)
(at_arm ?a - arm ?y - waypoint)
(position_loading ?a - arm)
(position_unloading ?a - arm)
```

```
(free_arm ?a - arm)
         (available_arm ?a - arm)
(free_object ?o - object)
(in_store ?o - object ?r - rover)
(in_arm ?o - object ?a - arm)
(equipped_for_soil_analysis ?r - rover)
(equipped_for_rock_analysis ?r - rover)
(at_soil_sample ?w - waypoint)
(at_rock_sample ?w - waypoint)
(have_soil_analysis ?r - rover ?w - waypoint)
(have_rock_analysis ?r - rover ?w - waypoint)
(at_lander ?l - lander ?w - waypoint)
(communicated_soil_data ?w - waypoint)
(communicated_rock_data ?w - waypoint)
(gate_charge_area ?cz - chargeZone ?w - waypoint ?d - direction)
(rover_in_charge ?r - rover ?cz - chargeZone)
(arm_in_charge ?a - arm ?cz - chargeZone)
)

(:action navigate
:parameters (?r - rover ?y - waypoint ?z - waypoint ?d - direction)
:precondition
(and
(can_traverse ?r ?y ?z ?d)
(available_rover ?r)
(position_rover ?r ?d)
(at_rover ?r ?y)
(visible ?y ?z)
)
:effect
```

```
(and
(not (at_rover ?r ?y))
(at_rover ?r ?z)
)
)

(:action turn
:parameters (?r - rover ?o - direction ?d - direction)
:precondition
(and
(available_rover ?r)
(position_rover ?r ?o)
)
:effect
(and
(not (position_rover ?r ?o))
(position_rover ?r ?d)
)
)

(:action pick_up
:parameters (?a - arm ?o - object)
:precondition
(and
(available_arm ?a)
(free_arm ?a)
(position_loading ?a)
(free_object ?o)
)
:effect
```

```
(and
(not (free_object ?o))
(not (free_arm ?a))
(in_arm ?o ?a)
)
)

(:action drop
:parameters (?a - arm ?r - rover ?o - object ?w - waypoint ?cz - chargeZone)
:precondition
(and
(available_rover ?r)
(available_arm ?a)
(position_unloading ?a)
(in_arm ?o ?a)
(have_store ?r)
(empty_store ?r)
(rover_in_charge ?r ?cz)
(arm_in_charge ?a ?cz)
)
:effect
(and
(free_arm ?a)
(not (in_arm ?o ?a))
(not (empty_store ?r))
(in_store ?o ?r)
)
)

(:action move_loading
```

```
:parameters (?a - arm)
:precondition
(and
(available_arm ?a)
(position_loading ?a)
)
:effect
(and
(not (position_loading ?a))
(position_unloading ?a)
)
)

(:action move_unloading
:parameters (?a - arm)
:precondition
(and
(available_arm ?a)
(position_unloading ?a)
)
:effect
(and
(not (position_unloading ?a))
(position_loading ?a)
)
)

(:action sample_soil
:parameters (?r - rover ?w - waypoint)
:precondition
```

```
(and
(at_rover ?r ?w)
(at_soil_sample ?w)
(equipped_for_soil_analysis ?r)
(have_data_store ?r)
)
:effect
(and
(have_soil_analysis ?r ?w)
(not (at_soil_sample ?w))
)
)

(:action sample_rock
:parameters (?r - rover ?w - waypoint)
:precondition
(and
(at_rover ?r ?w)
(at_rock_sample ?w)
(equipped_for_rock_analysis ?r)
(have_data_store ?r)
)
:effect
(and
(have_rock_analysis ?r ?w)
(not (at_rock_sample ?w))
)
)

(:action enter_charge_area
```

```
:parameters (?r - rover ?w - waypoint ?d - direction ?cz - chargeZone)
:precondition
(and
(at_rover ?r ?w)
(position_rover ?r ?d)
(available_rover ?r)
(gate_charge_area ?cz ?w ?d)
)
:effect
(and
(rover_in_charge ?r ?cz)
(not (at_rover ?r ?w))
)
)

(:action exit_charge_area
:parameters (?r - rover ?w - waypoint ?d - direction ?cz - chargeZone)
:precondition
(and
(rover_in_charge ?r ?cz)
(position_rover ?r ?d)
(gate_charge_area ?cz ?w ?d)
(available_rover ?r)
)
:effect
(and
(not (rover_in_charge ?r ?cz))
(at_rover ?r ?w)
)
)
```

```
(:action communicate_soil_data
:parameters (?r - rover ?l - lander ?p - waypoint ?x - waypoint ?y - waypoint)
:precondition
(and
(at_rover ?r ?x)
(at_lander ?l ?y)
(have_soil_analysis ?r ?p)
(visible ?x ?y)
(available_rover ?r)
)
:effect
(and
(communicated_soil_data ?p)
)
)

(:action communicate_rock_data
:parameters (?r - rover ?l - lander ?p - waypoint ?x - waypoint ?y - waypoint)
:precondition
(and
(at_rover ?r ?x)
(at_lander ?l ?y)
(have_soil_analysis ?r ?p)
(visible ?x ?y)
(available_rover ?r)
)
:effect
(and
(communicated_rock_data ?p)
```

```
)
)
)
```

## D.2. Experimentos

A continuación se presenta el código en lenguaje PDDL, para cada uno de los experimentos descritos en el capítulo 4 de este documento.

### D.2.1. Experimento 1

```
(define
(problem pfile1)
(:domain Rovers)
(:objects
rover0 - rover
waypoint0 waypoint1 waypoint2 waypoint3 waypoint4 waypoint5 - waypoint
east north south west - direction
)
(:init
(visible waypoint0 waypoint1)
(visible waypoint0 waypoint3)
(visible waypoint1 waypoint0)
(visible waypoint1 waypoint2)
(visible waypoint1 waypoint4)
(visible waypoint2 waypoint1)
(visible waypoint2 waypoint5)
(visible waypoint3 waypoint0)
(visible waypoint3 waypoint4)
(visible waypoint4 waypoint1)
(visible waypoint4 waypoint3)
```

```
(visible waypoint4 waypoint5)
(visible waypoint5 waypoint4)
(visible waypoint5 waypoint2)
(at_rover rover0 waypoint0)
(position_rover rover0 east)
(available_rover rover0)
(have_store rover0)
(have_data_store rover0)
(empty_store rover0)
(equipped_for_soil_analysis rover0)
(equipped_for_rock_analysis rover0)
(can_traverse rover0 waypoint0 waypoint1 east)
(can_traverse rover0 waypoint0 waypoint3 south)
(can_traverse rover0 waypoint1 waypoint0 west)
(can_traverse rover0 waypoint1 waypoint2 east)
(can_traverse rover0 waypoint1 waypoint4 south)
(can_traverse rover0 waypoint2 waypoint1 west)
(can_traverse rover0 waypoint2 waypoint5 south)
(can_traverse rover0 waypoint3 waypoint0 north)
(can_traverse rover0 waypoint3 waypoint4 east)
(can_traverse rover0 waypoint4 waypoint1 north)
(can_traverse rover0 waypoint4 waypoint3 west)
(can_traverse rover0 waypoint4 waypoint5 east)
(can_traverse rover0 waypoint5 waypoint2 north)
(can_traverse rover0 waypoint5 waypoint4 west)
(at_soil_sample waypoint4)
(at_rock_sample waypoint2)
)

(:goal
```

```
(and
(at_rover rover0 waypoint0)
(have_rock_analysis rover0 waypoint2)
(have_soil_analysis rover0 waypoint4)
)
)
)
```

### D.2.2. Experimento 2

```
(define
(problem pfile2)
(:domain Rovers)
(:objects
rover0 - rover
arm0 - arm
object0 - object
waypoint0 waypoint1 waypoint2 waypoint3 waypoint4 waypoint5 waypoint6 waypoint7 - w
east north south west - direction
)
(:init
(visible waypoint0 waypoint1)
(visible waypoint0 waypoint3)
(visible waypoint1 waypoint0)
(visible waypoint1 waypoint2)
(visible waypoint1 waypoint5)
(visible waypoint2 waypoint1)
(visible waypoint2 waypoint3)
(visible waypoint2 waypoint6)
(visible waypoint3 waypoint2)
```

```
(visible waypoint3 waypoint7)
(visible waypoint4 waypoint0)
(visible waypoint4 waypoint5)
(visible waypoint5 waypoint1)
(visible waypoint5 waypoint4)
(visible waypoint5 waypoint6)
(visible waypoint6 waypoint2)
(visible waypoint6 waypoint5)
(visible waypoint6 waypoint7)
(visible waypoint7 waypoint3)
(visible waypoint7 waypoint6)
(at_rover rover0 waypoint0)
(position_rover rover0 east)
(available_rover rover0)
(have_store rover0)
(empty_store rover0)
(have_data_store rover0)
(equipped_for_soil_analysis rover0)
(equipped_for_rock_analysis rover0)
(can_traverse rover0 waypoint0 waypoint1 east)
(can_traverse rover0 waypoint1 waypoint5 south)
(can_traverse rover0 waypoint1 waypoint0 west)
(can_traverse rover0 waypoint2 waypoint1 west)
(can_traverse rover0 waypoint2 waypoint3 east)
(can_traverse rover0 waypoint2 waypoint6 south)
(can_traverse rover0 waypoint3 waypoint2 west)
(can_traverse rover0 waypoint5 waypoint1 north)
(can_traverse rover0 waypoint5 waypoint6 east)
(can_traverse rover0 waypoint6 waypoint2 north)
(can_traverse rover0 waypoint6 waypoint5 west)
```

```
(can_traverse rover0 waypoint6 waypoint7 east)
(can_traverse rover0 waypoint7 waypoint3 north)
(can_traverse rover0 waypoint7 waypoint6 west)
(at_soil_sample waypoint5)
(at_rock_sample waypoint3)
)

(:goal
(and
(at_rover rover0 waypoint0)
(have_rock_analysis rover0 waypoint3)
(have_soil_analysis rover0 waypoint5)
)
)
)
```

### D.2.3. Experimento 3

```
(define
(problem pfile3)
(:domain Rovers)
(:objects
rover0 - rover
arm0 - arm
object0 - object
waypoint0 waypoint1 waypoint2 waypoint3 waypoint4 waypoint5 waypoint6 waypoint7 wa
east north south west - direction
charge0 - chargeZone
)
(:init
```

```
(visible waypoint0 waypoint1)
(visible waypoint0 waypoint3)
(visible waypoint1 waypoint0)
(visible waypoint1 waypoint2)
(visible waypoint1 waypoint5)
(visible waypoint2 waypoint1)
(visible waypoint2 waypoint3)
(visible waypoint2 waypoint6)
(visible waypoint3 waypoint2)
(visible waypoint3 waypoint7)
(visible waypoint4 waypoint0)
(visible waypoint4 waypoint5)
(visible waypoint5 waypoint1)
(visible waypoint5 waypoint4)
(visible waypoint5 waypoint6)
(visible waypoint6 waypoint2)
(visible waypoint6 waypoint5)
(visible waypoint6 waypoint7)
(visible waypoint6 waypoint8)
(visible waypoint7 waypoint3)
(visible waypoint7 waypoint6)
(visible waypoint7 waypoint9)
(visible waypoint8 waypoint6)
(visible waypoint8 waypoint9)
(visible waypoint9 waypoint7)
(visible waypoint9 waypoint8)
(at_rover rover0 waypoint4)
(position_rover rover0 east)
(available_rover rover0)
(have_store rover0)
```

```
(empty_store rover0)
(have_data_store rover0)
(equipped_for_soil_analysis rover0)
(equipped_for_rock_analysis rover0)
(can_traverse rover0 waypoint0 waypoint1 east)
(can_traverse rover0 waypoint1 waypoint0 west)
(can_traverse rover0 waypoint1 waypoint2 east)
(can_traverse rover0 waypoint1 waypoint5 south)
(can_traverse rover0 waypoint2 waypoint1 west)
(can_traverse rover0 waypoint2 waypoint3 east)
(can_traverse rover0 waypoint3 waypoint2 west)
(can_traverse rover0 waypoint3 waypoint7 south)
(can_traverse rover0 waypoint4 waypoint0 north)
(can_traverse rover0 waypoint4 waypoint5 east)
(can_traverse rover0 waypoint5 waypoint1 north)
(can_traverse rover0 waypoint5 waypoint4 west)
(can_traverse rover0 waypoint5 waypoint6 east)
(can_traverse rover0 waypoint6 waypoint5 west)
(can_traverse rover0 waypoint6 waypoint7 east)
(can_traverse rover0 waypoint6 waypoint8 south)
(can_traverse rover0 waypoint7 waypoint3 north)
(can_traverse rover0 waypoint7 waypoint6 west)
(can_traverse rover0 waypoint8 waypoint9 east)
(can_traverse rover0 waypoint9 waypoint7 north)
(at_arm arm0 waypoint9)
(gate_charge_area charge0 waypoint9 east)
(arm_in_charge arm0 charge0)
(position_loading arm0)
(free_arm arm0)
        (available_arm arm0)
```

```
(free_object object0)
(at_soil_sample waypoint3)
(at_soil_sample waypoint8)
(at_rock_sample waypoint1)
)

(:goal
(and
(at_rover rover0 waypoint0)
(in_store object0 rover0)
(position_loading arm0)
(have_rock_analysis rover0 waypoint1)
(have_soil_analysis rover0 waypoint3)
(have_soil_analysis rover0 waypoint8)
)
)
)
```

### D.2.4. Experimento 4

```
(define
(problem pfile4)
(:domain Rovers)
(:objects
rover0 - rover
arm0 - arm
object0 - object
waypoint0 waypoint1 waypoint2 waypoint3 waypoint4 waypoint5 waypoint6 waypoint7 wa
east north south west - direction
charge0 - chargeZone
```

```
)
(:init
(gate_charge_area charge0 waypoint18 east)
(visible waypoint0 waypoint1)
(visible waypoint1 waypoint0)
(visible waypoint1 waypoint3)
(visible waypoint2 waypoint3)
(visible waypoint3 waypoint1)
(visible waypoint3 waypoint2)
(visible waypoint3 waypoint5)
(visible waypoint4 waypoint9)
(visible waypoint5 waypoint3)
(visible waypoint5 waypoint10)
(visible waypoint6 waypoint11)
(visible waypoint6 waypoint7)
(visible waypoint7 waypoint8)
(visible waypoint8 waypoint13)
(visible waypoint9 waypoint4)
(visible waypoint9 waypoint10)
(visible waypoint9 waypoint14)
(visible waypoint10 waypoint5)
(visible waypoint10 waypoint11)
(visible waypoint10 waypoint15)
(visible waypoint11 waypoint6)
(visible waypoint11 waypoint10)
(visible waypoint11 waypoint12)
(visible waypoint11 waypoint16)
(visible waypoint12 waypoint11)
(visible waypoint12 waypoint13)
(visible waypoint13 waypoint8)
```

```
(visible waypoint13 waypoint12)
(visible waypoint13 waypoint18)
(visible waypoint14 waypoint9)
(visible waypoint14 waypoint15)
(visible waypoint15 waypoint10)
(visible waypoint15 waypoint14)
(visible waypoint16 waypoint17)
(visible waypoint17 waypoint18)
(visible waypoint18 waypoint13)
(at_rover rover0 waypoint2)
(position_rover rover0 east)
(available_rover rover0)
(have_store rover0)
(empty_store rover0)
(have_data_store rover0)
(equipped_for_soil_analysis rover0)
(equipped_for_rock_analysis rover0)
(can_traverse rover0 waypoint0 waypoint1 east)
(can_traverse rover0 waypoint1 waypoint0 west)
(can_traverse rover0 waypoint1 waypoint3 south)
(can_traverse rover0 waypoint2 waypoint3 east)
(can_traverse rover0 waypoint3 waypoint1 north)
(can_traverse rover0 waypoint3 waypoint2 west)
(can_traverse rover0 waypoint3 waypoint5 south)
(can_traverse rover0 waypoint4 waypoint9 south)
(can_traverse rover0 waypoint5 waypoint3 north)
(can_traverse rover0 waypoint5 waypoint10 south)
(can_traverse rover0 waypoint6 waypoint11 south)
(can_traverse rover0 waypoint6 waypoint7 east)
(can_traverse rover0 waypoint7 waypoint8 east)
```

```
(can_traverse rover0 waypoint8 waypoint13 south)
(can_traverse rover0 waypoint9 waypoint4 north)
(can_traverse rover0 waypoint9 waypoint10 east)
(can_traverse rover0 waypoint9 waypoint14 south)
(can_traverse rover0 waypoint10 waypoint5 north)
(can_traverse rover0 waypoint10 waypoint11 east)
(can_traverse rover0 waypoint10 waypoint15 south)
(can_traverse rover0 waypoint11 waypoint6 north)
(can_traverse rover0 waypoint11 waypoint10 west)
(can_traverse rover0 waypoint11 waypoint12 east)
(can_traverse rover0 waypoint11 waypoint16 south)
(can_traverse rover0 waypoint12 waypoint11 west)
(can_traverse rover0 waypoint12 waypoint13 east)
(can_traverse rover0 waypoint13 waypoint8 north)
(can_traverse rover0 waypoint13 waypoint12 west)
(can_traverse rover0 waypoint13 waypoint18 south)
(can_traverse rover0 waypoint14 waypoint9 north)
(can_traverse rover0 waypoint14 waypoint15 east)
(can_traverse rover0 waypoint15 waypoint10 north)
(can_traverse rover0 waypoint15 waypoint14 west)
(can_traverse rover0 waypoint16 waypoint17 east)
(can_traverse rover0 waypoint17 waypoint18 east)
(can_traverse rover0 waypoint18 waypoint13 north)
(at_arm arm0 waypoint18)
(arm_in_charge arm0 charge0)
(position_loading arm0)
(free_arm arm0)
        (available_arm arm0)
(free_object object0)
(at_rock_sample waypoint3)
```

```
(at_soil_sample waypoint2)
(at_rock_sample waypoint11)
(at_soil_sample waypoint16)
(at_rock_sample waypoint13)
(at_soil_sample waypoint4)
)
(:goal
(and
(at_rover rover0 waypoint0)
(have_rock_analysis rover0 waypoint3)
(have_soil_analysis rover0 waypoint2)
(have_rock_analysis rover0 waypoint13)
(have_soil_analysis rover0 waypoint16)
(in_store object0 rover0)
)
)
)
```

### D.2.5. Experimento 5

```
(define
(problem pfile5)
(:domain Rovers)
(:objects
rover0 - rover
arm0 - arm
object0 - object
waypoint0 waypoint1 waypoint2 waypoint3 waypoint4 waypoint5 waypoint6 waypoint7 wa
east north south west - direction
charge0 - chargeZone
```

```
)
(:init
(visible waypoint0 waypoint1)
(visible waypoint0 waypoint1)
(visible waypoint0 waypoint2)
(visible waypoint1 waypoint0)
(visible waypoint1 waypoint3)
(visible waypoint2 waypoint0)
(visible waypoint2 waypoint3)
(visible waypoint2 waypoint6)
(visible waypoint3 waypoint1)
(visible waypoint3 waypoint2)
(visible waypoint3 waypoint7)
(visible waypoint4 waypoint5)
(visible waypoint4 waypoint11)
(visible waypoint5 waypoint4)
(visible waypoint5 waypoint6)
(visible waypoint5 waypoint12)
(visible waypoint6 waypoint2)
(visible waypoint6 waypoint5)
(visible waypoint6 waypoint7)
(visible waypoint6 waypoint13)
(visible waypoint7 waypoint3)
(visible waypoint7 waypoint6)
(visible waypoint7 waypoint8)
(visible waypoint7 waypoint14)
(visible waypoint8 waypoint7)
(visible waypoint8 waypoint9)
(visible waypoint8 waypoint15)
(visible waypoint9 waypoint8)
```

```
(visible waypoint9 waypoint10)
(visible waypoint9 waypoint16)
(visible waypoint10 waypoint9)
(visible waypoint10 waypoint17)
(visible waypoint11 waypoint4)
(visible waypoint11 waypoint12)
(visible waypoint11 waypoint18)
(visible waypoint12 waypoint5)
(visible waypoint12 waypoint11)
(visible waypoint12 waypoint13)
(visible waypoint12 waypoint19)
(visible waypoint13 waypoint6)
(visible waypoint13 waypoint12)
(visible waypoint13 waypoint14)
(visible waypoint13 waypoint20)
(visible waypoint14 waypoint7)
(visible waypoint14 waypoint13)
(visible waypoint14 waypoint15)
(visible waypoint14 waypoint21)
(visible waypoint15 waypoint8)
(visible waypoint15 waypoint14)
(visible waypoint15 waypoint16)
(visible waypoint15 waypoint22)
(visible waypoint16 waypoint9)
(visible waypoint16 waypoint15)
(visible waypoint16 waypoint17)
(visible waypoint17 waypoint10)
(visible waypoint17 waypoint16)
(visible waypoint17 waypoint23)
(visible waypoint18 waypoint11)
```

```
(visible waypoint18 waypoint19)
(visible waypoint19 waypoint12)
(visible waypoint19 waypoint18)
(visible waypoint19 waypoint20)
(visible waypoint20 waypoint19)
(visible waypoint20 waypoint13)
(visible waypoint20 waypoint21)
(visible waypoint21 waypoint20)
(visible waypoint21 waypoint14)
(visible waypoint21 waypoint22)
(visible waypoint22 waypoint21)
(visible waypoint22 waypoint15)
(visible waypoint23 waypoint17)
(at_rover rover0 waypoint1)
(position_rover rover0 south)
(available_rover rover0)
(have_store rover0)
(empty_store rover0)
(have_data_store rover0)
(equipped_for_soil_analysis rover0)
(equipped_for_rock_analysis rover0)
(can_traverse rover0 waypoint0 waypoint1 east)
(can_traverse rover0 waypoint0 waypoint2 south)
(can_traverse rover0 waypoint1 waypoint0 west)
(can_traverse rover0 waypoint1 waypoint3 south)
(can_traverse rover0 waypoint2 waypoint0 north)
(can_traverse rover0 waypoint3 waypoint2 north)
(can_traverse rover0 waypoint3 waypoint7 south)
(can_traverse rover0 waypoint4 waypoint11 south)
(can_traverse rover0 waypoint5 waypoint6 east)
```

```
(can_traverse rover0 waypoint6 waypoint5 west)
(can_traverse rover0 waypoint6 waypoint13 south)
(can_traverse rover0 waypoint6 waypoint7 east)
(can_traverse rover0 waypoint7 waypoint3 north)
(can_traverse rover0 waypoint7 waypoint14 south)
(can_traverse rover0 waypoint8 waypoint7 west)
(can_traverse rover0 waypoint8 waypoint9 east)
(can_traverse rover0 waypoint8 waypoint15 south)
(can_traverse rover0 waypoint9 waypoint10 east)
(can_traverse rover0 waypoint10 waypoint17 south)
(can_traverse rover0 waypoint11 waypoint4 north)
(can_traverse rover0 waypoint11 waypoint12 east)
(can_traverse rover0 waypoint12 waypoint11 west)
(can_traverse rover0 waypoint12 waypoint19 south)
(can_traverse rover0 waypoint13 waypoint6 north)
(can_traverse rover0 waypoint13 waypoint14 east)
(can_traverse rover0 waypoint13 waypoint20 south)
(can_traverse rover0 waypoint14 waypoint7 north)
(can_traverse rover0 waypoint14 waypoint13 west)
(can_traverse rover0 waypoint15 waypoint8 north)
(can_traverse rover0 waypoint15 waypoint16 east)
(can_traverse rover0 waypoint16 waypoint15 west)
(can_traverse rover0 waypoint16 waypoint17 east)
(can_traverse rover0 waypoint17 waypoint16 west)
(can_traverse rover0 waypoint17 waypoint10 north)
(can_traverse rover0 waypoint17 waypoint23 south)
(can_traverse rover0 waypoint18 waypoint19 east)
(can_traverse rover0 waypoint19 waypoint20 east)
(can_traverse rover0 waypoint19 waypoint12 north)
(can_traverse rover0 waypoint20 waypoint13 north)
```

```
(can_traverse rover0 waypoint20 waypoint19 west)
(can_traverse rover0 waypoint20 waypoint21 east)
(can_traverse rover0 waypoint21 waypoint20 west)
(can_traverse rover0 waypoint21 waypoint22 east)
(can_traverse rover0 waypoint22 waypoint15 north)
(can_traverse rover0 waypoint22 waypoint21 west)
(can_traverse rover0 waypoint23 waypoint17 north)
(at_arm arm0 waypoint22)
(gate_charge_area charge0 waypoint22 east)
(arm_in_charge arm0 charge0)
(position_loading arm0)
(free_arm arm0)
        (available_arm arm0)
(free_object object0)
(at_rock_sample waypoint2)
(at_soil_sample waypoint3)
(at_rock_sample waypoint6)
(at_soil_sample waypoint10)
(at_rock_sample waypoint4)
(at_soil_sample waypoint15)
(at_rock_sample waypoint18)
(at_soil_sample waypoint23)
)
(:goal
(and
(at_rover rover0 waypoint23)
(have_rock_analysis rover0 waypoint2)
(have_soil_analysis rover0 waypoint3)
(have_soil_analysis rover0 waypoint23)
(have_soil_analysis rover0 waypoint15)
```

```
(have_rock_analysis rover0 waypoint4)
(have_soil_analysis rover0 waypoint10)
(in_store object0 rover0)
)
)
)
```

# Apéndice E

# Entorno de ejecución

En este apéndice se realiza una descripción del proceso a seguir para poder desplegar la arquitectura de control en una CPU. En primer lugar se realiza una descripción de los elementos del fichero de configuración y a continuación se describen todos los pasos necesario para desplegar la arquitectura.

## E.1. Fichero de configuración

El fichero de configuración, especifica la información necesaria para poder ejecutar el sistema. En este fichero se almacena toda la información referente a los directorios en los cuales se encontrarán el resto de ficheros, la localización del sistema de planificación que será utilizado, el directorio de almacenamiento de los ficheros de temporales, o incluso información referente a la velocidad de los robots, la distancia entre dos puntos del dominio, etc. En la figura E.1 se presenta un ejemplo de un fichero de configuración de los utilizados en este proyecto. En este ejemplo no se presentan todos los elementos que pueden ser incluidos en el fichero de configuración del sistema de control por lo que a continuación se realiza una descripción de todos los parámetros.

- FilesDir: Define la ruta relativa donde se almacenarán los distintos ficheros de gestión de definición de las distintas gramáticas utilizadas por el sistema.

```
<?xml version='1.0' encoding='utf-8'?>
<SettingsFile>
        <Settings>
                <Setting Name="FilesDir" Type="System.String">
                        <Value>files\</Value>
                </Setting>
                <Setting Name="ProblemsDir" Type="System.String">
                        <Value>sayphi\files\</Value>
                </Setting>
                <Setting Name="PlannersDir" Type="System.String">
                        <Value>c:\sayphi\</Value>
                </Setting>
                <Setting Name="DomainFile" Type="System.String">
                        <Value>domain.pddl</Value>
                </Setting>
                <Setting Name="ProblemFile" Type="System.String">
                        <Value>pfile6.pddl</Value>
                </Setting>
                <Setting Name="GrammarFile" Type="System.String">
                        <Value>grammar.cgt</Value>
                </Setting>
                <Setting Name="size" Type="System.Double">
                        <Value>700</Value>
                </Setting>
        </Settings>
</SettingsFile>
```

Figura E.1: Comunicación entre servicios y dispositivos

- ProblemsDir: Define la ruta relativa en el cual se encontrarán almacenados los ficheros de definición del dominio y los problemas.

- PlannersDir: Define la ruta absoluta donde se encuentra los archivos de ejecución del planificador.

- DomainFile: Define el nombre del fichero de definición del dominio del problema. Debe ser incluida la extensión del archivo.

- ProblemFile: Define el nombre del fichero del problema que va a ser resuelto por el sistema. Este fichero debe encontrarse almacenado en el directorio definido en el parámetro ProblemsDir.

- TempFile: Define el nombre del fichero que sera utilizado para generar el

fichero de problema temporal para el proceso de replanificación.

- GrammarFile: Define el nombre del fichero en el que se encuentran almacenadas las tablas del analizador de lenguajes para el procesado del fichero de definición del dominio. Este debe encontrarse en el directorio definido en el documento FilesDir.
- GrammarDomainFile: Define el nombre del fichero en el que se encuentran almacenadas las tablas del analizador de lenguajes para el procesado del fichero de definición del problema. Este debe encontrarse en el directorio definido en el documento FilesDir.
- Size: Define el tamaño de los *waypoints* del problema en centímetros. Todos los waypoints tiene el mismo tamaño.
- chargeZoneSize: Define el tamaño de la zona de carga en centímetros, donde se encuentran los robot de tipo excavador.
- MaxSpeed: Define la velocidad máxima en mm/s a la que se debe mover el robot.
- MaxPlannerTime: Define el tiempo máximo que empleará el planificador para resolver el problema, se expresa en segundos.

## E.2. Ejecución

En este apartado se realiza una descripción de los pasos que deben seguirse para configurar una CPU, en la cual pueda ser ejecutado el sistema de control. Además se incluye información referente a ciertos elementos que deben ser configurados en MRDS para poder ejecutar de forma correcta el sistema desarrollado y descrito en este documento..

1. Instalar Windows XP SP3. Es posible ejecutar el sistema de control desarrollado sobre windows vista o windows 7, pero sobre cada uno de ellos

surgieron algunos problemas con la ejecución de algunas de las funcionalidades de MRDS. Por lo que mi recomendación es utilizar el sistema operativo Windows XP SP3.

2. Instalar .Net framework 3.0o superior.

3. Instalar Microsoft Visual Studio 2008 o superior, este paso es opcional ya que esta herramienta sólo es necesaria para facilitar la compilación de las distintas librerías y servicios del sistema de control, pero es posible realizar la compilación por línea de comandos.

4. Instalar Microsoft Robotic Developer Studio 2008 R2 o superior.

5. Instalar el planificador Sayphi.

6. Tras la instalación de MRDS, es necesario configurar las opciones de seguridad de ejecución de los servicios, siendo la opción más sencilla, la desactivación de los protocolos de seguridad. Para poder definir los protocolos de seguridad de MRDS, es necesario crear un fichero de configuración XML, denominado **SecuritySettings.xml**. Este fichero debe ser almacenado en la carpeta **store** del directorio de instalación de MRDS, a continuación en la figura E.2, se presenta la estructura de un fichero de configuración en el cual se desactivan todos los protocolos de seguridad.

```
<?xml version="1.0"?>
<SecuritySettings
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns="http://schemas.microsoft.com/robotics/2008/02/security.html">
        <AuthenticationRequired>false</AuthenticationRequired>
        <OnlySignedAssemblies>false</OnlySignedAssemblies>
        <Contracts />
        <Roles />
</SecuritySettings>
```

Figura E.2: Ejemplo de fichero de seguridad.

7. Mover el código fuente del sistema de control al directorio de instalación de MRDS.

8. A continuación es necesario ejecutar el comando de migración, para que se reestructuren las rutas de los distintos servicios y librerías que utiliza el sistema de control. En la figura D.2 se presenta un ejemplo de la ejecución del comando de migración sobre el sistema de control que ha sido desarrollado para este proyecto.

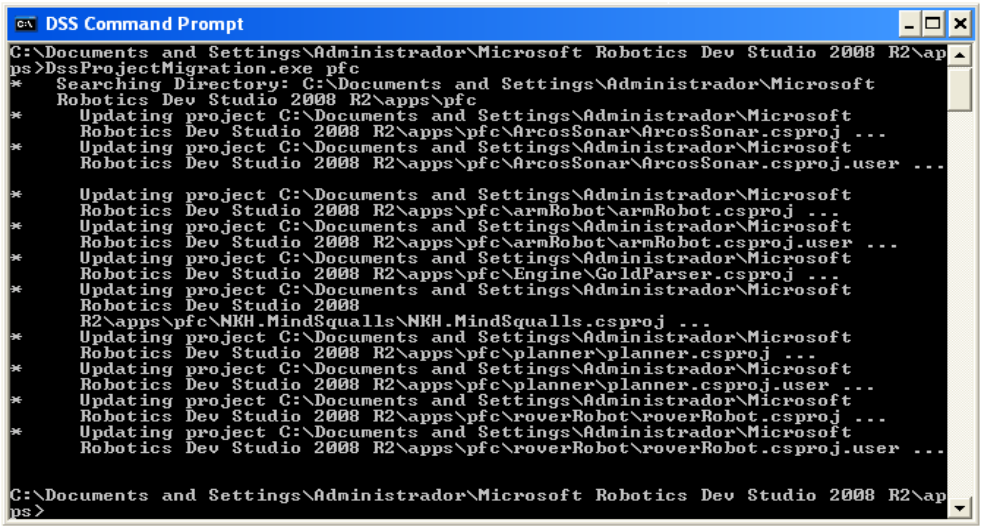


Figura E.3: Ejemplo de ejecución del comando de migración.

9. Realizar la compilación de todas las librerías y servicios del sistema de control.

10. Modificar las opciones del fichero de configuración del sistema de control, indicando las rutas en las que se encontrarán el fichero de definición del dominio, los ficheros de los problemas, la ruta de instalación del planificador y las distintas opciones del sistema.

11. Realizar la conexión con el brazo robot, mediante el sistema de comunicaciones basado en bluetooth.

12. Por último realizar la ejecución del sistema de control, bien desde el entorno gráfico que ofrece visual studio, o mediante linea de comando a través del comando de ejecución, cuya sintaxis se encuentra descrita en el anexo A de este documento. A continuación, en la figura E.3, se presenta un ejemplo de la ejecución del sistema de control.

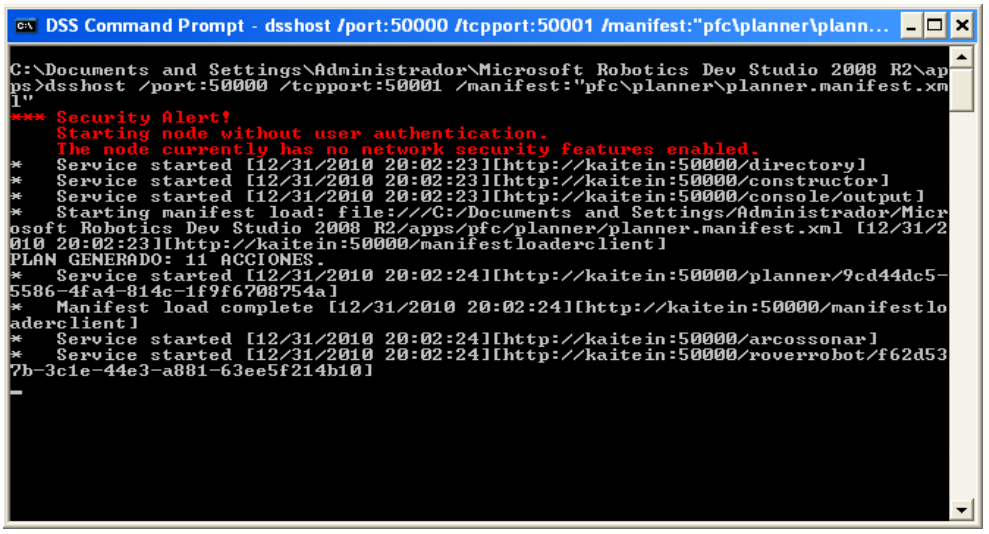


Figura E.4: Ejemplo de ejecución del sistema de control.

# Apéndice F

# Glosario de Términos

- .NET: Plataforma de desarrollo de Microsoft, enfocada a la construcción de programas para sistemas Windows. Permite la realización de aplicaciones en los lenguajes Visual C++, Visual Basic, C# y ASP.

- GANTT: Es un tipo de diagrama en el que se pueden ver fácilmente las diferentes tareas a realizar dentro de un proyecto, la duración, las dependencias entre las tareas y los recursos necesarios para realizar cada una de ellas.

- XML: Acrónimo de eXtensible Markup Language o Lenguaje de Marcado Extensible. Es un lenguaje etiquetado que permite crear documentos estructurados.

- Planificador: Es una herramienta software utilizada en el área de la planificación automática para la resolución de problemas, mediante la utilización en la mayor parte de los casos de algoritmos de búsqueda; una descripción del estado inicial del mundo; un conjunto de metas u objetivos a conseguir y un conjunto de acciones que pueden ser aplicadas sobre el entorno.

- GPS: El GPS (Global Positioning System) es un sistema global de navegación por satélite que permite determinar la posición de un objeto con una precisión de metros de error, es posible disminuir el error a pocos centímetros

utilizando GPSP (GPS Posicional).

- Compás: Un compás o brújula es un sensor magnético, utilizado en robótica para medir o detectar la orientación de un robot.

- Bumper: Un bumper es una sensor de presión, utilizado frecuentemente en la robótica para detectar los choques que se producen contra otros objetos.

- Giroscopio: Un giroscopio es un dispositivo mecánico formado esencialmente por un cuerpo con simetría de rotación que gira alrededor de su eje de simetría. Es utilizado en la robótica para conocer la inclinación del robot con respecto a un eje. Por ejemplo nos permitiría conocer la inclinación del terreno por el cual se mueve el robot.

- Sónar: El sónar (Sound Navigation And Ranging) es una técnica que usa la propagación de ondas de sonido para poder detectar objetos que se encuentran en la trayectoria de las ondas. En la robótica frecuentemente es utilizado como sistema de detección de objetos con el fin de evitarlos y obtener información para la construcción de mapas.

- Actuador: Los actuadores son dispositivos capaces de generar una fuerza a partir de líquidos , energía eléctrica o energía gaseosa. El actuador recibe la orden de un regulador o controlador y da una salida necesaria para activar a un elemento final de control, como por ejemplo, una válvula.

- Sensor: Un sensor es un dispositivo capaz de detectar magnitudes físicas o químicas, llamadas variables de instrumentación, y transformarlas en variables eléctricas.

- Puerto de comunicación: Un puerto de comunicaciones es una interfaz a través de la cual diferentes tipos de datos pueden ser enviados y recibidos. Dicha interfaz puede ser de tipo físico, o de tipo lógico, como la empleada por los protocolos de comunicación.

- Autómata finito: Un autómata finito (AF) o máquina de estado finita, es un modelo matemático que acepta expresiones de un lenguaje definido sobre un alfabeto. Su estructura esta compuesta por un conjunto finito de estados, que representan elementos del lenguaje, relacionados entre sí mediante transiciones.

- Servicio Web: Un servicio web (en inglés, Web service) es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones.

- SOAP: SOAP (según sus siglas en inglés, Simple Object Access Protocol) es un protocolo estándar de comunicaciones que define cómo dos objetos de diferentes procesos o sistemas pueden comunicarse por medio del intercambio de información codificada mediante de XML.

- Clausura de Kleene: En lógica matemática, la clausura de Kleene es una operación unaria que se aplica sobre un conjunto de cadenas de caracteres, símbolos o caracteres (alfabeto). Esta representa el conjunto de las cadenas que pueden ser formadas mediante cualquier número de cadenas del conjunto inicial, normalmente permitiendo la aparición de cadenas repetidas o concatenadas. A nivel matemático, la aplicación de esta clausura sobre un conjunto C, se representa como $C^*$.

- Notación Backus-Naur: Es una meta-sintaxis utilizada para definir gramáticas libres de contexto, por ejemplo aquellas que se utilizan para definir la sintaxis de los lenguajes de programación, los sistemas de comandos y los protocolos de comunicación, e incluso en algunos casos para representar fragmentos de las gramáticas del lenguaje natural.

- Gramática libre de contexto: En lingüística e informática, una gramática libre de contexto, es aquella donde cada regla de producción es de la forma $A \rightarrow a$. Donde A es un símbolo de tipo no terminal y a es una cadena de terminales y/o no terminales. El concepto de libre de contexto se refiere a

que el símbolo no terminal A, puede ser sustituido por a sin tener encuentra el contexto en el cual se produzca la sustitución.

- Sayphi: Es un planificador de propósito general, desarrollado por el grupo PLG de la Universidad Carlos III de Madrid, que implementa varias técnicas. Además, ofrece diferentes técnicas de aprendizaje.

# Bibliografía


[1] M. Ai-Chang, J. Bresina, L. Charest, A. Chase, J. C. jung Hsu, A. Jonsson, B. Kanefsky, P. Morris, K. Rajan, J. Yglesias, B. G. Chafin, W. C. Dias, and P. F. Maldague, "Mapgen: Mixed-initiative planning and scheduling for the mars exploration rover mission," *EEE Intelligent Systems*, pp. 8–12, 2004.

[2] M. de la Asunción, L. Castillo, J. Fdez-Olivares, O. García-Pérez, A. González, and F. Palao, "Siadex: an interactive artificial intelligence planner for decision support and training in forest fire fighting," *Artificial Intelligence Communications*, no. 18, 2005.

[3] I. Asimov, *Runaround*. Science-fiction story, 1942.

[4] K. Capek, *Rossum's Universal Robots*. Science-fiction story, 1921.

[5] S. Technology, "Shakey, the robot." Website, 2006. `http://www.sri.com/about/timeline/shakey.html`.

[6] G. Ernst and A. Newell, "Gps: A case study in generality and problem solving," *ACM Monograph Series. Academic Press, New York, NY, United State of America*, 1969.

[7] S. Russell, S. D., and P. R., "Provably bounded optimal agents," *Proceedings 13th International Joint Conference on Artificial Intelligence*, pp. 338–344, 1993.

[8] R. Fikes and N. Nilsson, "Strips: a new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, no. 2, pp. 189–208, 1971.

[9] E. Gat, M. Slack, D. P. Miller, and R. J. Firby, "Path planning and execution monitoring for a planetary rover," *Proceding of the IEEE International Conf. on Robotics and Automation*, pp. 22–25, 1990.

[10] R. A. Brooks, "A robust layered control system for a mobile robots.," *IEEE Journal of Robotics and Automation*, no. E2, pp. 14–23, 1986.

[11] R. A. Brooks, "Inteligence without representation.," *Artificial Intelligence*, no. 47, pp. 139–159, 1991.

[12] R. A. Brooks, "Integrated systems based on behaviors.," *SIGART Bulletin*, no. 2, pp. 46–50, 1991.

[13] R. P. Bonasso, J. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. G. Slack, "Experiences with an architecture for intelligent, reactive agents.," *Journal of Experimental & Theoritical Artificial Intelligence*, pp. 237–256, 1997.

[14] R. Burke and B. Blumberg, "Using an ethologically-inspired model to learn apparent temporal causality for planning in synthetic creatures.," *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 326–333, 2002.

[15] E. Gat, "Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real world mobile robots," *Proceedings of the National Conference on Artificial Intelligence, San Jose, California.*, pp. 809–815, 1992.

[16] E. Gat, "Reliable goal-directed reactive control of autonomous mobile robots," *Ph.D. Dissertation, Computer Science and Applications, Virginia Polytechnic Institute*, 1991.

[17] R. Arkin, "Motor schema-based mobile robot navigation," *International Journal of Robotics Research*, no. 4, pp. 92–112, 1989.

[18] D. Rumelhart, "Schemata: the building blocks of cognition.," *Theoretical Issues in Reading Comprehension, ed. Rand J. Spiro, Bertram C. Bruce, and William F. Brewer. Hillsdale, NJ: Lawrence Erlbaum*, 1980.

[19] T. Bylander, "The computational complexity of propositional strips planning," *Artificial Intelligence*, no. 69, pp. 165–204, 1994.

[20] E. P. D. Pednault, "Adl and the state-transition model of action.," *Journal of Logic and Computation*, no. 4, pp. 467–512, 1994.

[21] M. Fox and D. Long, "Pddl2.1: An extension to pddl for expressing temporal planning domains," *Journal of Artificial Intelligence Research*, pp. 61–124, 2003.

[22] H. A. Kautz and B. Selman, "Planning as satisfiability," *Conference on Artificial Intelligence*, pp. 359–363, 1992.

[23] A. Blum and M. Furst, "Fast planning through planning graph analysis," *Conference on Artificial Intelligence*, no. 90, pp. 281–300, 1997.

[24] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Science and Cybernetics*, no. 4, pp. 100–107, 1968.

[25] J. Hoffmann and B. Nebel, "The ff planning system: Fast plan generation through heuristic search," *Journal of Artificial Intelligence Research*, no. 14, pp. 253–302, 2001.

[26] T. de la Rosa, A. García Olaya, and D. Borrajo, "Using cases utility for heuristic planning improvement," *Proceedings of the 7th International Conference on Case-Based Reasoning, Belfast Northern Ireland, Springer-Verlag*, 2007.

[27] K. Johns and T. Taylor, *Microsoft Robotic Developer Studio*. Wiley Publishing, Inc., 2008.

[28] S. Morgan, *Programming Microsoft® Robotics Studio*. Microsoft Press., 2006.

[29] "Mindsquall library." Website. `http://www.mindsquall.net`.

[30] D. Cook, "Gold parser library." Website. `http://www.devincook.com/goldparser`.

[31] E. Alfonseca Cubero, M. Alfonseca Cubero, and R. Moriyón Salomón, *Teoría de autómatas y lenguajes formales*. McGraw-Hill, 2007.